International Journal of Pattern Recognition
and Artificial Intelligence
Vol. 31, No. 6 (2017) 1755009 (33 pages)

DOI: 10.1142/S0218001417550096

World Scientific
www.worldscientific.com

Information Based Universal Feature Extraction in Shallow Networks

Mohammad Amiri* and Rüdiger Brause
J. W.Goethe University, Frankfurt a.M., Germany
**amiri@fias.uni-frankfurt.de*

Received 2 October 2015
Accepted 4 November 2016
Published 26 January 2017

In many real-world image based pattern recognition tasks, the extraction and usage of task-relevant features are the most crucial part of the diagnosis. In the standard approach, either the features are given by common sense like edges or corners in image analysis, or they are directly determined by expertise. They mostly remain task-specific, although human may learn the life time, and use different features too, although same features do help in recognition. It seems that a universal feature set exists, but it is not yet systematically found. In our contribution, we try to find such a universal image feature set that is valuable for most image related tasks. We trained a shallow neural network for recognition of natural and non-natural object images before different backgrounds, including pure texture and handwritten digits, using a Shannon information-based algorithm and learning constraints. In this context, the goal was to extract those features that give the most valuable information for classification of the visual objects, hand-written digits and texture datasets by a one layer network and then classify them by a second layer. This will give a good start and performance for all other image learning tasks, implementing a transfer learning approach. As result, in our case we found that we could indeed extract unique features which are valid in all three different kinds of tasks. They give classification results that are about as good as the results reported by the corresponding literature for the specialized systems, or even better ones.

Keywords: Machine vision; universal feature extraction; information theory; transfer learning; extreme learning.

1. Introduction

Humans have sought to extract information from imagery ever since the first photographic images were acquired.[29] The most useful basic components are called *features*. Feature extraction and representation are crucial steps for object recognition. One issue is the effective identification of important features in images, and the other one is extracting them. It is a difficult task to obtain a prior knowledge of what kind of information is required from the image, even when you know the image domain. Feature extraction is a type of dimension reduction that

efficiently represents interesting parts of an image as a compact feature vector with less data. Features are functions of original measurements that are useful for classification and/or pattern recognition. In other words, feature extraction of images is the process of defining a set of image characteristics, which represent most efficiently or significantly the information that is important for analysis and classification; much of the information in the data set may be of little value for discrimination.

There have been many attempts to solve this problem. Dong[35] presents a review on image feature extraction and representation techniques. In his view, there are three feature representations: global, block-based, and region-based features. Chow *et al.*[4] proposed an image classification approach through a tree-structured feature set. In this approach, they combined both the global and the local features through the root and the child nodes. Tsai and Lin[38] compared global, block-based, and region-based features and their combinations by using a standard classifier over 30 categories. However, it is not clear whether these features are important or not. All those feature definitions seem to arbitrary subjective, not guided by the task specification itself.

Now, what is the basic task? Let us shortly sketch the problem. A picture is worth a thousand words: As human beings, we are able to tell a story from a picture based on what we see, using our background knowledge. However, the first step for any computer vision program is to extract efficiently visual features and build models from them, and after this, use context knowledge.[35] So, how to extract image low-level visual features and what kind of features are extracted plays a crucial role in various tasks of image processing? Color, shape and spatial relations are the main features human beings, as well as computers use to recognize images,[1,8,18,24,25,33,36,39] and most of the image processing and machine vision systems use those features. However, there is a general agreement that the tools available for analysis of images are not sufficient. Additionally, it is still a challenging problem in computer vision how to extract *universal features* that reflect the fundamental substance of images as complete as possible.

In this context, we might ask: Do *universal features* in images exist such that by using them we are able to efficiently recognize any unknown object? Is it necessary to extract new special features for any new object recognition tasks? How about using existing features from other tasks? Is it possible to use extracted feature of a specific task for other tasks? Are there some general features in natural and nonnatural images which can also be used for specific object recognition? For example, can we use extracted features of natural images also for handwritten digit classification? Very little research attention has been paid to these problems in the last decades. Some people used the concept of *transfer learning* to reuse the knowledge taken from one classification problem for similar problems. Dan *et al.*[3] used the knowledge of Latin digits classification for Latin uppercase letters. Raina *et al.*[30] also used a similar paradigm which is *self-taught learning*, or transfer learning from unlabeled data to

use the knowledge of some nonlabeled data for supervised classification of groups of animals with limited number of images.

This paper proposes a new information based approach and tries to give some answers. Therefore, in this paper, we focus on the extraction of very general features that can be useful for object recognition and classification, implemented by a neural network of only two layers.

The rest of the paper is organized as follows. Section 2 elaborates the definition of universal features and describes the proposed method for their extraction. In Sec. 3, we present the implementation of the method by neural networks, and in Sec. 4, some experimental results of this algorithm are shown. Finally, some important conclusions and future potential research directions are shown in Sec. 5.

2. Universal Feature Extraction

In this section, we will develop the notation of universal features. How can we show that a feature is universal or not? One criterion is its applicability: a universal feature has to be effective in all applications ever existed and yet to come up. Unfortunately, there is no practical way to prove this. Instead, we will first define the features by theoretical considerations and then show their effectiveness.

Alternatively, we may not need to prove that a certain feature is universal; it rather means that it is not specific to any particular application. For example, textures are common in many computer vision applications which means for many texture features to have a multipurpose nature. For applications of different nature, texture feature extracting algorithms may determine what morphological particularities are typical in all those applications. The algorithms addressing these types are then also practicable in this scenario.

In this contribution, for task specification we will focus on the question: What kind of features are the best for classifying objects? It is well known that the best strategy for classification is the Bayes decision criterion[9]: given an image x, choose that class ω_k which has the highest conditional probability of occurrence,

$$\omega_k = \arg\max_{\omega_i} P(\omega_i|x). \tag{1}$$

Unfortunately, we do not know the conditional probabilities. Instead, we have to estimate them.

Let us assume that we observe pictures x containing an object. Additionally, a teacher will tell us with the decision L if the object is present: $L = 1$ indicates *yes* and $L = 0$ means *no*. Therefore, the observation set consists of pairs (x, L) and the best classification is the one which maximizes the probability $P(L|x)$. Now, instead of using the whole picture, only a small set of features $h_1, \ldots, h_n$ extracted from x by a function $h(x)$ should be sufficient for detecting the object. How can we find it? Let us first consider just one feature h. This means, that the probability of the correct decision for the presence of object $P(L|x)$ should be as close to $P(L|h)$ as possible.

Since the probability for correct classification is based on the conditional probabilities, the distance between the two probability distributions can be seen as a measure for the classification quality, implementing the Bayes decision. It is well known that the Kullback–Leibler distance,

$$D(P(L|x)||P(L|h)) = \sum_L P(L|x) \log \frac{P(L|x)}{P(L|h)}, \tag{2}$$

becomes zero if and only if the two probability distributions become equal.[6] It implements the difference

$$\begin{aligned} \sum_L P(L|x) \log \frac{P(L|x)}{P(L|h)} &= \sum_L P(L|x) \log P(L|h) - \sum_L P(L|x) \log P(L|x) \\ &= H_L(x) - H_L(h) \end{aligned} \tag{3}$$

between the estimated Shannon information $H_L(h)$ and the observed information $H_L(x)$ of the image pattern x for the teacher classification decision L.

Now, we have a problem: since $h(x)$ is an unknown function, we do not know $P(L|h)$. Instead, we can estimate it by a function $g(L|h)$ which does depend on the decision L, but is indeed a function of h only. Therefore, we note it by $g_L(h)$. Nevertheless, if we maintain $0 < g < 1$ the Kullback–Leibler distance will still become zero if the two probability distributions become equal. Therefore, we might use the expected distance as an objective function R for setting up the unknown function.

$$\begin{aligned} R &= \sum_x P(x) D(P(L|x)||g_L(h)) \\ &= \sum_x \sum_L P(x) P(L|x) \log \frac{P(L|x)}{g_L(h)} \\ &= \sum_x \sum_L P(L,x) \log \frac{P(L|x)}{g_L(h)} \\ &= \sum_x \sum_L P(L,x) \log P(L|x) - \sum_x \sum_L P(L,x) \log g_L(h). \end{aligned} \tag{4}$$

The objective function is composed by two additive terms. The first term does not depend on the unknown function g, remaining constant when changing g. Therefore, for *minimizing* R, it suffices to *maximize* the new risk function

$$R(g,h) = \sum_x \sum_L P(L,x) \log g_L(h) = \langle \log g_L(h(x)) \rangle. \tag{5}$$

The expectation $\langle . \rangle$ is computed over all values of x and L. This is also covered by the uniformly distributed M observations $(x(i), L(i))$, where $i = 1, \ldots, M$ by

$$R(g,h) = \frac{1}{M} \sum_{i=1}^{M} \log g_L(h(x(i))). \tag{6}$$

In our observation set, each $x(i)$ is accompanied by the teacher decision $L(i) \in \{0, 1\}$. For $L(i) = 1$, the feature should be present to show the presence of the object. Assuming the probability $g_L(h)$ for $L = 1$ is $g(h)$, then for the second case $L = 0$, the probability must be $(1 - g(h))$. Therefore, the term $\log g(h)$ in the objective function can be written as:

$$\log g_L(h) = L(i) \log g(h) + (1 - L(i)) \log(1 - g(h)), \tag{7}$$

and the objective function becomes:

$$R(g, h) = \frac{1}{M} \sum_{i=1}^{M} L(i) \log g(h) + (1 - L(i)) \log(1 - g(h)). \tag{8}$$

This function is well known as *maximum likelihood objective function.*[27] It should be mentioned that for N independent objects, the probabilities of the multiple output $\mathbf{h} = (h_1, \ldots, h_N)$ factorize,

$$g_L(\mathbf{h}) = \prod_{k=1}^{N} g_{Lk}(h_k), \tag{9}$$

and the log probability becomes by Eq. (7)

$$\log g_L(\mathbf{h}) = \sum_{k=1}^{N} L_k(i) \log g_k(h_k) + (1 - L_k(i) \log(1 - g_k(h_k))). \tag{10}$$

Thus, our objective risk function forms a sum over all single risks,

$$R(g, \mathbf{h}) = \frac{1}{M} \sum_{i=1}^{M} \sum_{k=1}^{N} L_k(i) \log g_k(h_k) + (1 - L_k(i) \log(1 - g_k(h_k))). \tag{11}$$

Now, how can we obtain the unknown functions g and h? Let us assume that we use parameterized functions, i.e. the necessary information for extracting and using the features are stored in a finite set of parameters. For m features, we assume m extraction functions $h_i(x)$, each one containing n parameters,

$$h_j(x) = h_j(\mathbf{u}, x) \quad \text{with } \mathbf{u} = (u_1, \ldots, u_n).$$

The object detection function $g(\mathbf{y})$ is determined by s parameters

$$g(\mathbf{h}(\mathbf{u}), \mathbf{w}) \quad \text{with } \mathbf{h} = (h_1, \ldots, h_m) \quad \text{and } \mathbf{w} = (w_1, \ldots, w_s).$$

Thus, the task of determining the universal features becomes a task of determining the appropriate parameters of the unknown functions.

3. Learning the Feature Extraction

In this section, we will describe our approach for extracting the universal features by minimizing the objective function. Unfortunately, the desired solution is problem dependent, i.e. it depends on our observation set. One common approach for

minimizing an objective function, if there is no analytic solution, is the stepwise iteration of an approximation expression, a so-called *learning algorithm*, using the observations as *training set*.

As learning algorithm for the parameters $\mathbf{w}$, we might use the well-known stochastic gradient ascend for maximizing R,

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \gamma(t)\frac{\partial R(g(\mathbf{w}))}{\partial \mathbf{w}}, \tag{12}$$

which does not use the expectation value over M samples of the objective function R,

$$R(g,\mathbf{h}) = \frac{1}{M}\sum_{i=1}^{M} L(i)\log g(\mathbf{h}) + (1-L(i))\log(1-g(\mathbf{h})) = \frac{1}{M}\sum_{i=1}^{M} R_i(g,\mathbf{h}), \tag{13}$$

but its stochastic version for the ith sample pairs $(x(i), L(i))$,

$$R_i(g,\mathbf{h}) = L(i)\log g(\mathbf{w}, h(\mathbf{u}, i)) + (1-(L(i)))\log(1-g(\mathbf{w},(h(\mathbf{u},i)))). \tag{14}$$

For further computations, let us drop the index i from the notation, since the formulas should be applied to all pairs $(x(i), L(i))$ of the training set. The gradient of the stochastic objective function becomes

$$\begin{aligned}
\frac{\partial}{\partial \mathbf{w}} R(\mathbf{w},\mathbf{u}) &= \frac{\partial}{\partial \mathbf{w}} L\log g(\mathbf{w}) + (1-L)\log(1-g(\mathbf{w})) \\
&= \frac{L}{g(\mathbf{w})}\frac{\partial g(\mathbf{w},y)}{\partial \mathbf{w}} - \frac{1-L}{1-g(\mathbf{w})}\frac{\partial g(\mathbf{w})}{\partial \mathbf{w}} \\
&= \left[\frac{L}{g} - \frac{1-L}{1-g}\right]\frac{\partial g(\mathbf{w})}{\partial \mathbf{w}} \\
&= \left(\frac{L(1-g) - g(1-L)}{g(1-g)}\right)\frac{\partial g(\mathbf{w})}{\partial \mathbf{w}} \\
&= \left(\frac{L-g}{g(1-g)}\right)\frac{\partial g(\mathbf{w})}{\partial \mathbf{w}}.
\end{aligned} \tag{15}$$

For the second set of parameters $\mathbf{u}$, we proceed analogously. Here, our learning algorithm is:

$$\mathbf{u}(t+1) = \mathbf{u}(t) + \gamma(t)\frac{\partial R(g(\mathbf{u}))}{\partial \mathbf{u}}, \tag{16}$$

and the gradient becomes:

$$\frac{\partial}{\partial \mathbf{u}} R(\mathbf{w},\mathbf{u}) = \left(\frac{L-g}{g(1-g)}\right)\frac{\partial g(\mathbf{w}, h(\mathbf{u}))}{\partial \mathbf{u}}. \tag{17}$$

For estimating the unknown function g and the parameters $\mathbf{w}$, we learn the parameters by iteratively analyzing the data.

3.1. *The neural net for extracting one feature*

Now, we have to choose the kind of functions $g(.)$ and $h(.)$ to use here. It is well known that all continuous functions can be approximated sufficiently well by two-layers networks using sigma neurons and sigmoid functions S as output functions.[16] Therefore, choosing our approximation functions like this will not limit our approach in any way. With the image input described by a pixel tuple $\mathbf{x}$, we might choose as extraction function a squashing function with an affine argument

$$h_j(\mathbf{u}, \mathbf{x}) = S(z) \quad \text{with } z = \mathrm{u}^T\mathrm{x},$$

and as object detection function for one object, we choose the Fermi function:

$$g(\mathbf{w}, \mathbf{h}) = S_F(v) \quad \text{with } S_F(v) = \frac{1}{1 + \exp(-v)} \quad \text{and} \quad v = \mathrm{w}^T\mathrm{h}.$$

This can be interpreted as having a first layer of formal neurons, implementing sigma neurons and squashing function $h(\mathbf{u}, \mathbf{x})$, and a second layer, implementing the object detection function $g(\mathbf{h}, \mathbf{w})$. In Fig. 1, the two-layer architecture is shown with N output units, each one detecting a different object.

Now, to obtain the desired iteration equations, the learning rules, we use the standard back-propagation approach for our risk function and compute the necessary derivatives.

In our approach and our learning rules, we have the properties $0 \leq g \leq 1$ and $0 \leq L \leq 1$. This is covered by the choice of the Fermi function $S_F(v) = \frac{1}{1+\exp(-v)}$ as squashing function of the output layer with $\dim(\mathbf{w}) = \dim(\mathbf{h}) = m$ and the

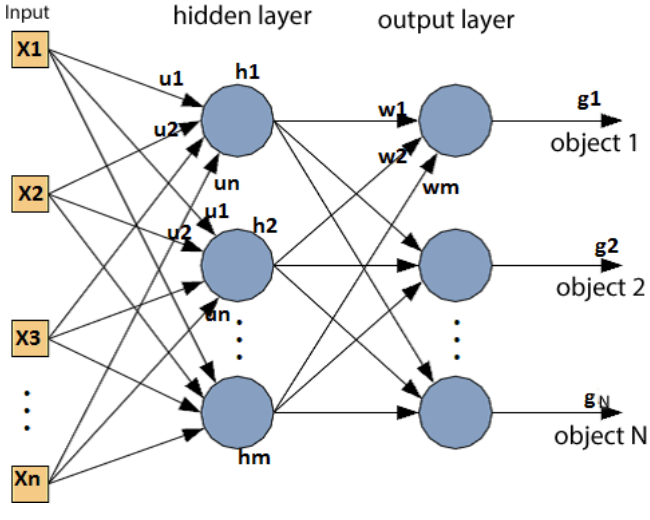

Fig. 1. The network architecture for function approximation.

derivative:

$$\frac{\partial g}{\partial v} = \frac{\partial S_F(v)}{\partial v} = \frac{\partial}{\partial v}(1+e^{-v})^{-1} = (1+e^{-v})^{-2}e^{-v} = \frac{1+e^{-v}-1}{(1+e^{-v})(1+e^{-v})},$$
$$= g(1-g), \tag{18}$$

For the first layer, the *hidden layer*, we get,

$$h_j(\mathbf{u},\mathbf{x}) = S_t(z) = \text{e.g. } \tanh(z) \text{ with } z = \mathbf{u}^T\mathbf{x}.$$

Therefore, the derivatives in Eqs. (12), (16) become

$$\begin{aligned}\frac{\partial}{\partial \mathbf{w}} R(\mathbf{w},\mathbf{u}) &= \left(\frac{L-g}{g(1-g)}\right)\frac{\partial g(\mathbf{w},h(\mathbf{u}))}{\partial \mathbf{w}} \\ &= \left(\frac{L-g}{g(1-g)}\right)\frac{\partial g}{\partial v}\frac{\partial v}{\partial \mathbf{w}} \\ &= \left(\frac{L-g}{g(1-g)}\right)g(1-g)\mathbf{h} = -(g-L)\mathbf{h},\end{aligned} \tag{19}$$

and

$$\begin{aligned}\frac{\partial}{\partial \mathbf{u}} R(\mathbf{w},\mathbf{u}) &= \left(\frac{L-g}{g(1-g)}\right)\frac{\partial g(\mathbf{w},h(\mathbf{u}))}{\partial \mathbf{u}} \\ &= \left(\frac{L-g}{g(1-g)}\right)\frac{\partial g}{\partial v}\frac{\partial v}{\partial \mathbf{u}} \\ &= \left(\frac{L-g}{g(1-g)}\right)g(1-g)\frac{\partial v}{\partial \mathbf{u}} = -(g-L)\frac{\partial v}{\partial \mathbf{u}}.\end{aligned} \tag{20}$$

The sth term of the vector $\frac{\partial v}{\partial \mathbf{u}}$ is

$$\frac{\partial v}{\partial u_s} = \sum_{j=1}^{m} w_j \frac{\partial h_j}{\partial u_s} = \sum_{j=1}^{m} w_j S'(z_j)\frac{\partial z_j}{\partial u_s} = \sum_{j=1}^{m} w_j S'(z_j)x_s. \tag{21}$$

By this, our learning Eqs. (12), (16) become:

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \gamma_1(t)(g-L)\mathbf{h}, \tag{22}$$

$$\mathbf{u}(t+1) = \mathbf{u}(t) - \gamma_2(t)(g-L)\sum_{j=1}^{m} w_j S'(z_j)\mathbf{x}, \tag{23}$$

with e.g. $S'_t(z_j) = 1 - h_j^2$.

Now, for learning we assume several important restrictions:

- Each extraction function h_j covers a different part of input $\mathbf{x}$, i.e. it has a unique receptive field and is not completely overlapping with other fields, see Fig. 2. This means, that the tuple of input pixels $\mathbf{x}$ is different for each extraction unit j, denoted by $\mathbf{x}_j$.

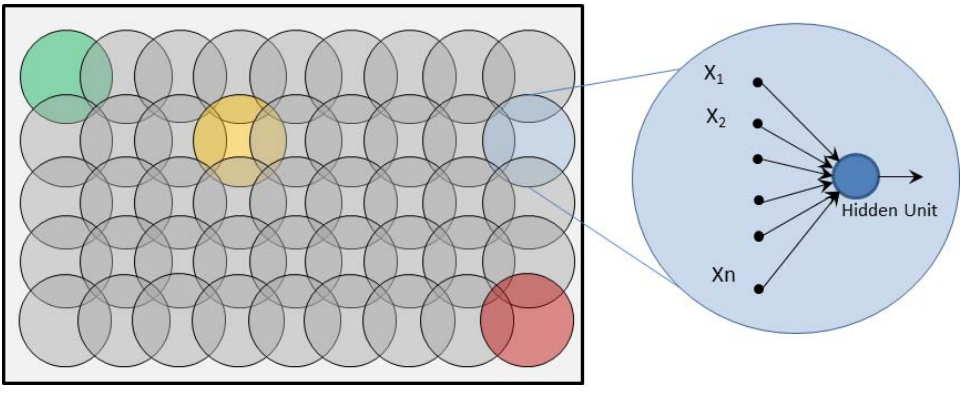

Fig. 2. The receptive field patch extraction from an image.

- The object should be recognized everywhere on the image. Therefore, in order to train only the statistics and avoid overfitting, we put the constraint that the parameters $\mathbf{u}$ of each extraction function are the same ones, i.e. all hidden neurons share the same k weights.
- There can be more than one object present, i.e. N ones which should be recognized independently. Therefore, we assume not one, but N functions g_k, i.e. N output units.
- Training a weight will also result in training neighboring weights by a certain degree.

An important decision in this network is that we use the weight sharing idea in the feature extraction layer. Using weight sharing has two advantages: First, it reduces the number of parameters for learning, and second, all neurons learn to detect the same features, although, their receptive fields are located at different positions in the input image.

The number of output neurons depends on the number of classes (sets) that we need or how many sets we want for classification. Therefore, the weights $\mathbf{w}$ in the last layer are not shared and specific for the detected classes. In Fig. 3, the overall architecture is shown.

We use the first layer (U) as feature extractor and the second layer (W) as classifier layer. Since all outputs $g_k(\mathbf{w}_k, \mathbf{h})$, can be computed independently from each other, the stochastic gradient learning rule does not change much.

For the kth output unit, we get by Eq. (22),

$$\mathbf{w}_k(t+1) = \mathbf{w}_k(t) - \gamma_1(t)(g_k - L_k)\mathbf{h}, \tag{24}$$

and Eq. (23) becomes by all N output units:

$$\mathbf{u}(t+1) = \mathbf{u}(t) - \gamma_2(t) \sum_{k=1}^{N} (g_k - L_k) \sum_{j=1}^{m} w_{kj} S'(z_j) \mathbf{x}_j, \tag{25}$$

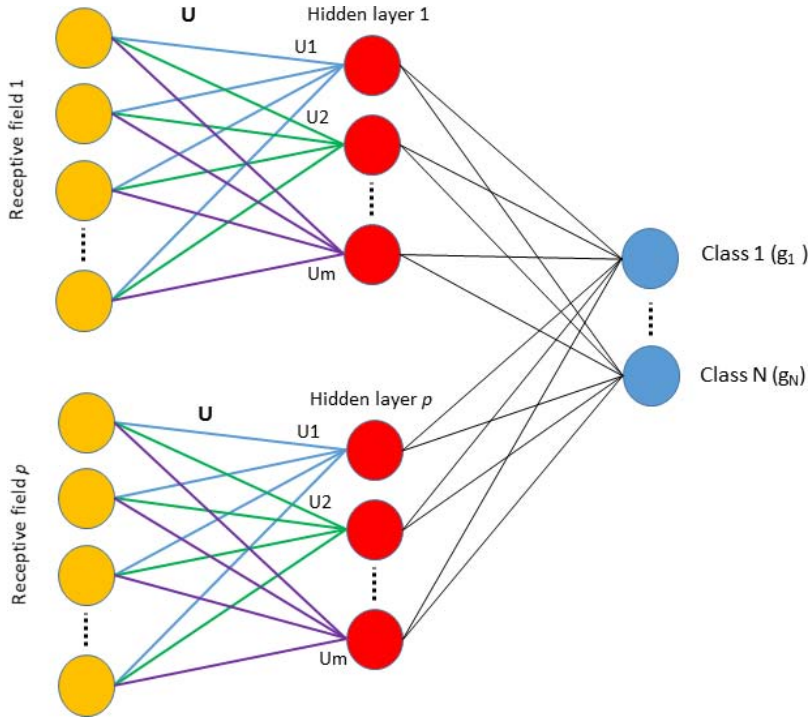

Fig. 3. The architecture of the two layers neural network for universal feature extraction.

and each component of $\mathbf{u}(u_{rs})$ in two-dimensional view of weights has a little effect on its neighbors u_{mn} by

$$u_{mn}(t+1) = u_{mn}(t) + \aleph(m,n,r,s)u_{rs}(t+1), \tag{26}$$

where

$$\aleph(m,n,r,s) = e^{-((r-m)^2+(s-n)^2)/2\sigma^2} \tag{27}$$

is the neighborhood function and variable σ in this equation is related to neighborhood radius. It is updated for each unit j differently. Here, m, n, r, s are the indices of the $\mathbf{u}$ in two-dimensional view of it. In Fig. 4, this is shown.

The input samples are no longer treated similarly by the extraction units $h_j(x)$, but they are grouped into subsets. Each unit j processes only a subset x_j. The input samples can be arranged in different manners. On the left-hand side of Fig. 4, the samples are arranged in a two-dimensional manner, e.g. like pixels of an image. The one-dimensional case is shown on the right-hand side of Fig. 4, e.g. for a speech signal with $k = 5$.

As you can also see in Fig. 4, we extract several patches from each image and use them as inputs for the network. The number of patches that can be extracted from an image depends on some factors, e.g. the size of a patch, the size of the image and the

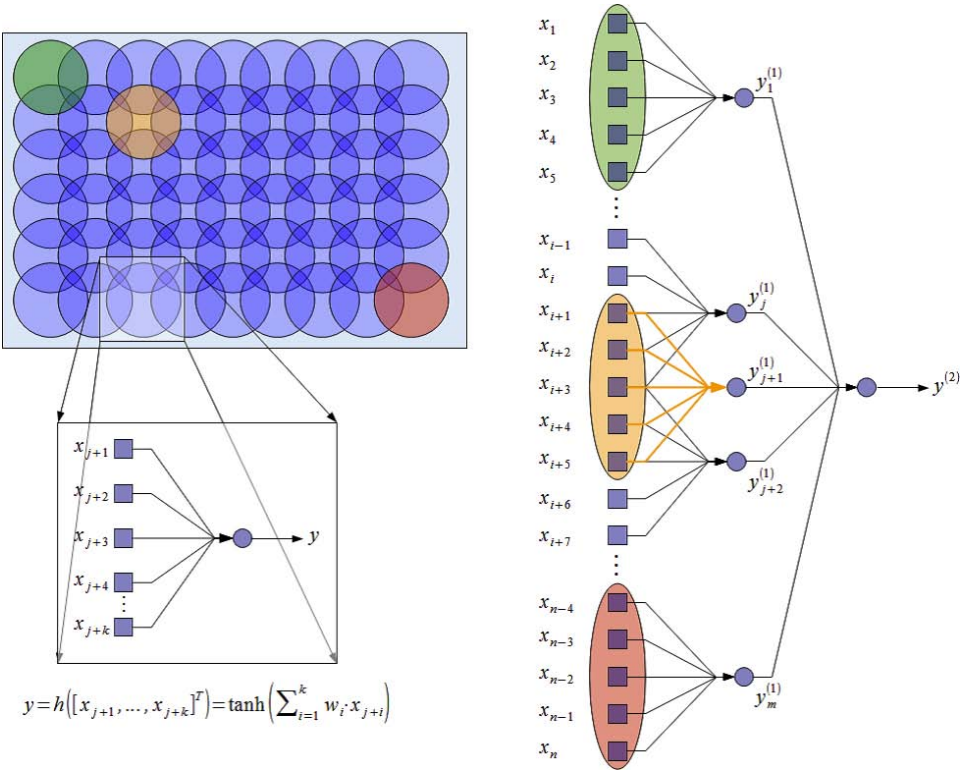

Fig. 4. The input samples covered by the first layer by two-dimensional overlapping receptive fields (left) or one-dimensional overlapping receptive fields (right) (from Ref. 15).

number of pixels shared between two neighbor patches. For instance, the number of rectangular patches which can be taken from an image with 60×80 pixels and a patch size of 9×9 sharing three pixels is 108. Please note that, we extract square patches instead of circular ones because it is computationally more feasible.

There are still some open questions for this kind of architecture:

- What is the best size of a receptive field (patch)?
- What is the optimum number of hidden units?

We will discuss these questions in later sections presenting some experimental results. It is clear that, by increasing the size of the image, we need more receptive fields and more parameters in the subsequent layer. Instead, it might be better to increase the receptive field size for covering the image by a smaller number of fields.

3.2. *Extracting several features*

Our feature extraction analysis of the previous section only covers just one feature in each receptive field, the most important one. How do we get additional, helpful

features? Let us assume that in each receptive field we extract not only one feature, but r ones. Then, each extraction result h_j of receptive field j has several components,

$$h_j = (h_1(\mathbf{u}_1, \mathbf{x}_j), \ldots h_r(\mathbf{u}_r, \mathbf{x}_j))^T \quad \text{with } h_i(\mathbf{u}_i, \mathbf{x}_j) = S(z_{ij}), z_{ij} = \mathbf{u}_j^T \mathbf{x}_j. \tag{28}$$

The corresponding network architecture is shown in Fig. 5. On the left-hand side, we see the two-dimensional input sample image covering. On the right-hand side, the corresponding one-dimensional receptive fields are shown. The activity of the second layer, the object detection, will not change except the fact that for each output unit, the number of inputs becomes mr instead of m.

$$g(\mathbf{w}, \mathbf{h}) = S_F(v) \quad \text{and } v = \mathbf{w}^T \mathbf{h} \text{ and } \dim(\mathbf{w}) = \dim(\mathbf{h}) = mr. \tag{29}$$

Certainly, the learning equations change with the additional features. Equation (22) has now mr components, and Eq. (23) becomes for the sth feature:

$$\mathbf{u}_s(t+1) = \mathbf{u}_s(t) - \gamma_2(t)(g - L)\frac{\partial v}{\partial \mathbf{u}_s}. \tag{30}$$

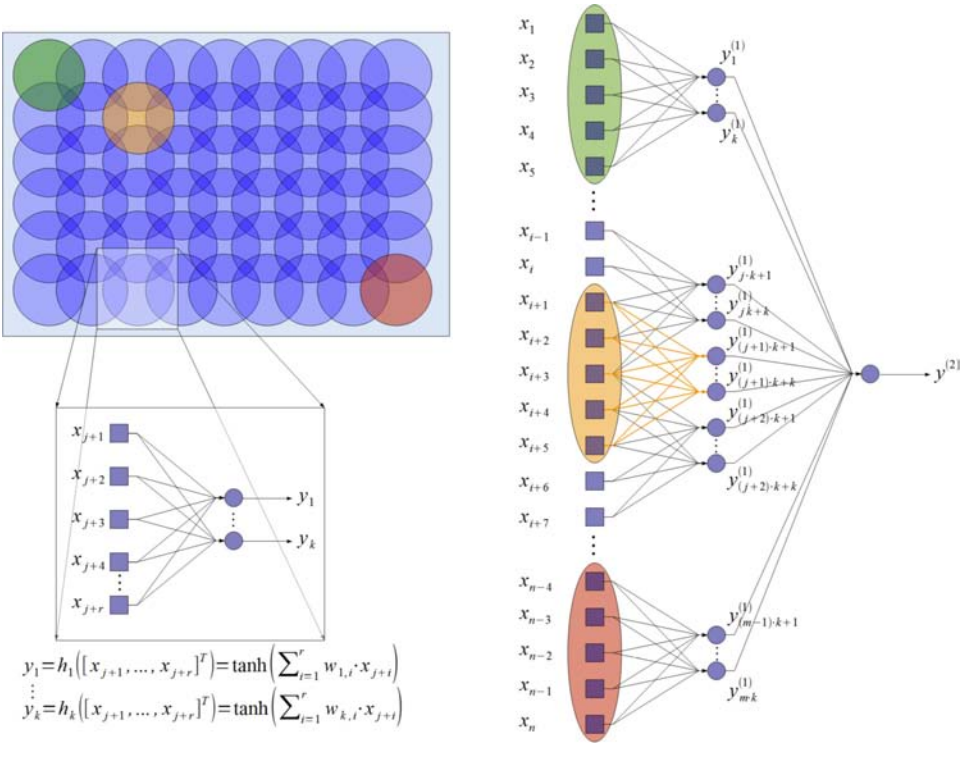

Fig. 5. The extraction of multiple features (from Ref. 15).

With the activity of

$$v = \mathbf{w}^T \mathbf{h} = \sum_{p=1}^{rm} w_p h_p = \sum_{i=1}^{r} \sum_{j=1}^{m} w_{ij} h_{ij} + w_0, \tag{31}$$

containing the ith feature of the jth receptive field, we get the derivative of

$$\frac{\partial v}{\partial \mathbf{u}_s} = \frac{\partial}{\partial \mathbf{u}_s} \sum_{i=1}^{r} \sum_{j=1}^{m} w_{ij} h_{ij} = \sum_{i=1}^{r} \sum_{j=1}^{m} w_{ij} \frac{\partial}{\partial \mathbf{u}_s} h_{ij}(\mathbf{u}_i, \mathbf{x}_j). \tag{32}$$

Since the ith feature extraction function $h_{ij}(\mathbf{u}_i, \mathbf{x}_j)$ depend only on the ith parameter vector $\mathbf{u}_i$, we get zero for all terms where $i \neq s$

$$\frac{\partial v}{\partial \mathbf{u}_s} = \sum_{i=1}^{r} \sum_{j=1}^{m} w_{ij} \frac{\partial}{\partial \mathbf{u}_s} h_{ij}(\mathbf{u}_i, \mathbf{x}_j) = \sum_{j=1}^{m} w_{sj} S'(z_{sj}) \mathbf{x}_j, \tag{33}$$

and our learning equation becomes

$$\mathbf{u}_s(t+1) = \mathbf{u}_s(t) - \gamma_2(t)(g - L) \sum_{j=1}^{m} w_{sj} S'(z_{sj}) \mathbf{x}_j, \tag{34}$$

using k inputs x_{kj} at each receptive field j, obtaining the feature $y_{sj} = S_t(z_{sj}) =$ e.g. $\tanh(\mathbf{u}_s^T \mathbf{x}_j)$ with $S'(z_{sj}) = 1 - \tanh^2(z_{sj})$.

For N outputs, Eq. (22) changes to

$$\mathbf{w}_k(t+1) = \mathbf{w}_k(t) - \gamma_1(t)(g_k - L_k)\mathbf{h}, \tag{35}$$

and Eq. (34) is determined by all N output units:

$$\mathbf{u}_s(t+1) = \mathbf{u}_s(t) - \gamma_2(t) \sum_{k=1}^{N} (g_k - L_k) \sum_{j=1}^{m} w_{ksj} S'(z_{sj}) \mathbf{x}_j, \tag{36}$$

and here also like in Eqs. (26) and (27) each component of $\mathbf{u}(u_{pq})$ in two-dimensional view of weights has a little effect on its neighbors u_{mn} by

$$u_{mn}(t+1) = u_{mn}(t) + \aleph(m, n, p, q) u_{pq}(t+1). \tag{37}$$

Now, there are r features learned by each of the receptive fields. How can we assume that they will be different although they have the same input and the same learning rules? The answer lies in the fact that all feature vectors $\mathbf{u}_s$ have different feedback from the second layer, depending on their own activity h_s. This leads to different learning behavior and different convergence states.

3.3. *Training and testing*

There are several training strategies which distinguish among different learning modes:

1. *Parallel training*: All weights w_{kij} and u_{sp} are updated after the whole net has determined its activity. This strategy is preferable, but suffers from strong convergence problems of the network parameter iteration.
2. *Sequential training*: We first train $\mathbf{u}_1$ and the corresponding w_{k1} until convergence, not using the other $\mathbf{u}_2, \ldots, \mathbf{u}_r$. Then, leaving $\mathbf{u}_1$ constant, we train $\mathbf{u}_2$ and the w_{k1} and w_{k2}, still not using the other $\mathbf{u}_3, \ldots, \mathbf{u}_r$. After this, we train $\mathbf{u}_3$ including $w_{k1}, \ldots, w_{k3}$ and so on, until all the other feature vectors are determined.
3. *Batch versus stochastic training*: Parallel or sequential training can be used as elements in a more comprehensive strategy, the use of batch offline or stochastic online learning. Let us show this for the proposed back-propagation scheme by the following nested loops of pseudo code for batch training.

In contrast to batch offline learning, stochastic online learning differs slightly. It does not take the average overall corrections, but use them instantly. Thus, each learning step is based on the previous one and not on the average overall patterns. For the training procedure, we might ask the following questions: Does the performance decrease with increasing system size k? Does the performance increase with increasing number f of features? The results of these different training procedures are different. The stochastic training converges faster, but has higher performance variations than the batch procedure.

4. Experimental Results

In this section, we report several results using the ideas and algorithms presented so far. First, we discuss the setup of the training procedure and some of the network parameters used. Then, the results of training and testing with different kind of images and parameters are reported.

4.1. *Network parameters*

To prepare the network for training, several decisions have to be taken before. First, let us discuss the general decisions which are taken for training and testing.

Activation functions: There are number of common activation functions in use for artificial neural networks e.g. tanh, the step function, Gaussian function for RBF nets, the *logistic sigmoid function* $f(x) = 1/(1 + e^{-x})$ or the *bipolar sigmoid function* $f(x) = (1 - e^{-x})/(1 + e^{-x})$. In some literature, e.g. Ref. 32, it is emphasized that, although selection of an activation function for a neural network or its node is an important task, other factors like the training algorithm, the network size or the

learning parameters are more vital for a proper training of the network. In Ref. 19, it has been shown that for general purpose *bipolar sigmoid*, *unipolar sigmoid* and *tanh* functions are better than others. In our case, we used the bipolar *tanh* activation function for the hidden layer units of the network and the unipolar sigmoid function for the output units of the second layer, because the output should show the amount of probability that an input object may be in a class. Therefore, the output function has to take values between zero and one.

Weight initialization: There are also many possible algorithms for initializing the weights for feed forward neural networks.[10,19,20] One method is the usual weight initialization: a uniform random initialization inside the interval $[-0.05, +0.05]$ or $[-0.01, +0.01]$. For large number of inputs the smaller random interval is preferred to avoid the saturation of the sigmoid functions. Random weight initialization is still the most popular method because of simplicity and comparable results with other methods.[10,12]

In Ref. 20, Kim proposed a minimum bound for the weight initialization. The initialization is still random, but satisfying a minimum value. In the equation, the minimum is the learning step used in the back-propagation training after initialization. In the reference, the initialization procedure is not clearly specified because there is just a lower bound and not an upper one,

$$\sqrt{(\gamma/n_{\text{input}})} < |w_{ij}|. \tag{38}$$

In Ref. 12, there is just a maximum bound for the weights and the initialization is still random, but satisfying the maximum,

$$|w_{ij}| < 2.4/n_{\text{input}}. \tag{39}$$

The variable n_{input} refers to the number of input units, w_{ij} refers to the weight between neuron j and input i and γ refers to the learning rate.

We used the method of random uniform distribution with interval $[-0.01, +0.01]$ for initializing the weights because it is simple and our experimental result showed that it performs better than the methods proposed in Refs. 12 and 20.

Learning rate: In all tests in training and test phase, the learning rate is $\gamma = 0.005$.

4.2. *Input data preparation*

Some object images are taken from the Amsterdam library of object images (ALOI) database.[26] ALOI is a color image collection of 1000 small objects, recorded for scientific purposes. In order to capture the sensory variations in object appearance, they systematically varied viewing angle, illumination angle, and illumination color for each object and additionally captured wide baseline stereo images. They recorded over a hundred images of each object, yielding a total of 110,250 images for the whole collection.[26] Objects can be characterized as natural (e.g. an apple or an orange) or artificial (e.g. a hat or a cup), see Figs. 6 and 7.

Algorithm 1. Algorithm of Batch Training (*offline learning*)

```
for all features f do
  for all units k, increasing system size do
    for all cross-validations p do
      reset weights w_kf and u_f for unit k and feature f
      for all iteration steps t do
        for all patterns i of the training set do
          compute the activity (w, u) of the current network layers
          compute the corrections Δw_kf and Δu_f for the k − th
          unit and the f − th feature
        end for i
        update the k − th weights w_kf and u_f for feature f
        compute the objective function R(training set), R(test set)
      end for t
      change training and test set
    end for p
    compute the average of R(.) of all training and test sets p for one
    system size k
  end for k
  compute R(.) of the full system size and one feature f
end for f
```

We placed the selected objects in the middle of some natural or artificial background images and used shifted variations of three pixels left, right, up or down, maximally. By this, we prepared six sets of data. For preprocessing the input images, we normalized each input pixel set $\mathbf{x}$ to zero mean and unit variance of all pixel values. It is also possible to normalize only the set of extracted patches instead of normalizing the whole image, but the result of image normalization was better. The size of the objects to recognize is 80×60 pixels. Objects in sets 8–11 in addition of having different viewing angle, illumination angle and illumination color, have also different scales (1, 2 and 2.5). These sets are multi-scale versions of set 1–4 and are designed for training and testing scale invariance. These objects are placed before different backgrounds. For example, Figs. 8 and 9 show different objects with multiple scales, illumination and viewing angles, placed before different background images.

Additionally, we used different kinds of datasets including MNIST handwritten digits, KTH-TIPS2 and UIUCTEX texture to validate that features are universal.

The MNIST database of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples. The digit images have 28×28 pixels.[27] Initially, before use, we normalized the size and centered it in a fixed-size image.

Algorithm 2. Algorithm of Stochastic Gradient (*online learning*)

```
for all features f do
  for all units k, increasing system size do
    for all cross-validations p do
      reset weights w_kf and u_f for unit k and feature f
      for all iteration steps t do
        for all patterns i of the training set do
          compute the activity (w, u) of the current network layers
          compute the corrections Δw_kf and Δu_f for the k − th
          unit and the f − th feature
          update the k − th weights w_kf and u_f for feature f
        end for i
        compute the objective function R(training set), R(test set)
      end for t
      change training and test set
    end for p
    compute the average of R(.) of all training and test sets p for one
    system size k
  end for k
  compute R(.) of the full system size and one feature f
end for f
```

The KTH-TIPS (Textures under varying Illumination, Pose and Scale) image database was created to extend the CUReT database in two directions, by providing variations in scale as well as pose and illumination, and by imaging other samples of a subset of its materials in different settings. The KTH-TIPS2 databases took this a step further by imaging 4 different samples of 11 materials, each under varying pose, illumination and scale.[28] The UIUC texture database features 25 texture classes, 40 samples each. All images are in grayscale JPG format, 640×480 pixels.[29]

Figures 10, 11 and 12 show some examples of these three datasets. These sets were chosen to represent unnatural objects. If features are *universal*, they have to represent efficiently also those objects. Examples of the resulting training and test objects are shown in Fig. 13.

The following Table 1 gives an overview of the composition of the different training and test sets. In set 2, objects are shifted a little from the center and they have different view and illuminations with many natural backgrounds. In set 4, the same objects are used as in set 2, but they use nonnatural backgrounds. In sets 1 and 3, objects are natural (like apples or potatoes) with the same backgrounds as in sets 2 and 4, respectively. Data sets 8–11 have multiple scales in addition to multiple

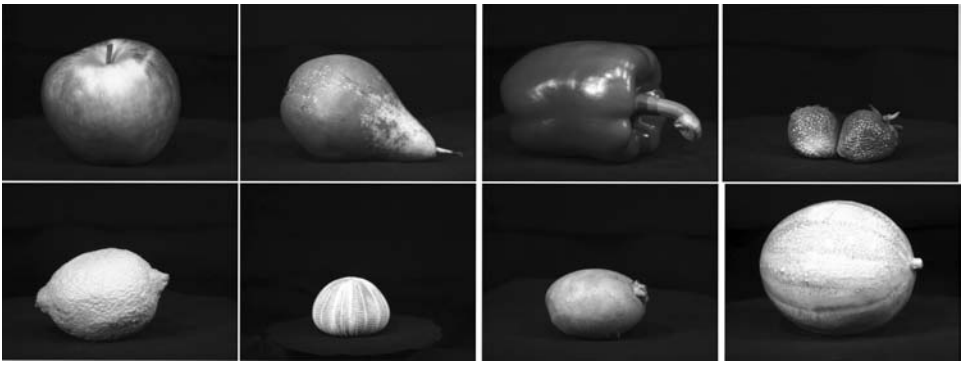

Fig. 6. Some examples of natural objects.

Fig. 7. Some examples of artificial objects.

views and illuminations and natural or nonnatural background, similar to sets 1–4 respectively.

4.3. *Does the network learn universal features?*

We trained the system with 40,000 input images in 10 different groups with 16 neurons in the hidden layer. Each group included one object with multiple

Fig. 8. Example objects located on multiple **natural** background.

Fig. 9. Example objects located on multiple **artificial** background.

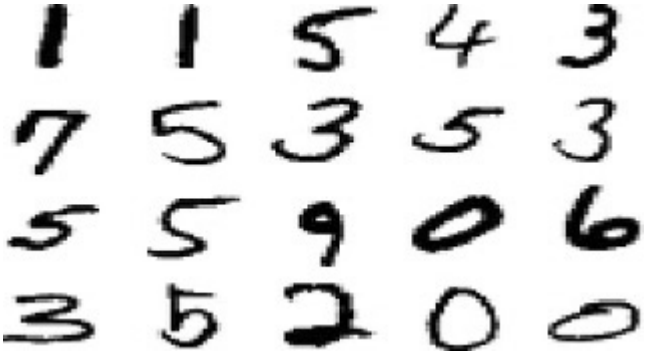

Fig. 10. Some examples of MNIST handwritten digits.

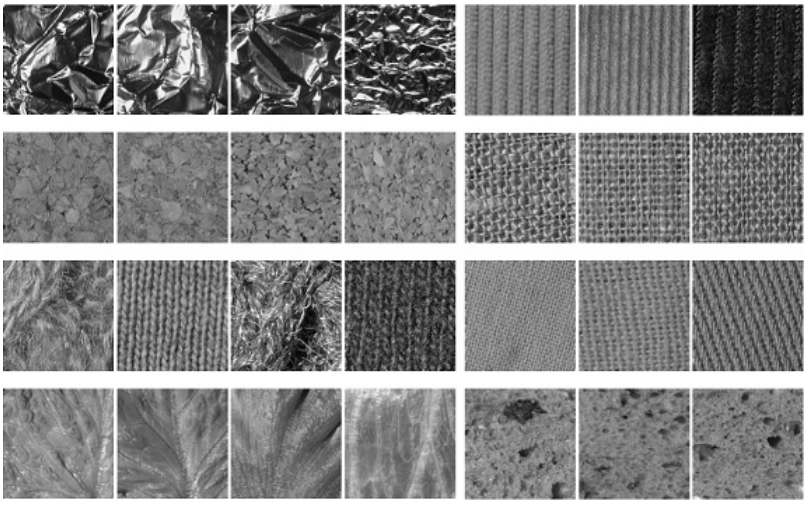

Fig. 11. Some examples of KTH-TIPS2 texture dataset.

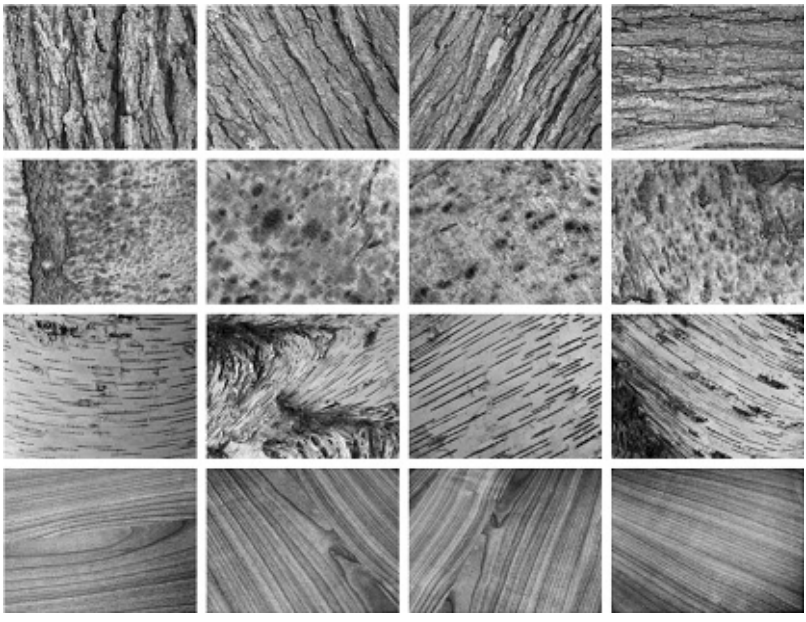

Fig. 12. Some examples of UIUC texture dataset.

Fig. 13. Image examples of the training and test sets.

Table 1. The composition of the training and test sets.

Set Label	Object	Background	Scale
Set 1	Natural	Natural	1
Set 2	Artificial	Natural	1
Set 3	Natural	Artificial	1
Set 4	Artificial	Artificial	1
Set 5	MNIST handwritten digits		1
Set 6	KTH texture		1
Set 7	UIUC texture		1
Set 8	Natural	Natural	1, 2, 2.5
Set 9	Artificial	Natural	1, 2, 2.5
Set 10	Natural	Artificial	1, 2, 2.5
Set 11	Artificial	Artificial	1, 2, 2.5

illuminations and view angles, placed in the middle of many background images, with a maximum of three pixel shift in right or left and up or down. The size of a receptive field was set to 9×9, and each receptive field shared three pixels with its neighbors. This value was determined experimentally; it gives better result than others. After convergence of the network, we fixed the value of the first layer (U), the features, and used it as feature extractor for further processing.

The weights of the second layer were trained separately to classify multiple objects with multiple backgrounds. After training, it could classify 10 groups (according to the objects in the images) of data set images. For evaluation, we used as classification accuracy

$$\text{Accuracy} = 0.5(\text{Prob}(PT) + \text{Prob}(NF)). \tag{40}$$

Please note that, for calculating the rate of accuracy, we had to record the positive (true) PT and negative (false) NF system classification decisions. If we just use the positive input PT rate to compute the accuracy rate, by changing the threshold value we can get better results. Therefore, for a fair comparison, we had to take both rates into account. In general, a ROC analysis have to be computed, but the averaged correct decision is sufficient for this application. For more information about ROC analysis, see Refs. 11 and 28. For computing the probabilities, we used the classification output of the neural network units. Because the output of the units is between zero and one, to assign an object to a class, we selected the maximum value of the output in accordance to the Bayes classification rule. Thus, the object is the member of a class with the maximum output value

$$\text{choose } C_i \text{ if } g_i = \max(g_j) \quad j = 1, \ldots, N. \tag{41}$$

It is also possible to use a Softmax policy to assign an object to a specific class. For instance, after using set 2 as training set to learn the features, the test revealed that with 95.70% accuracy set 1 was correctly classified, and with 97.50% accuracy set 4.

Table 2. Universal feature learning: test results.

	Test Set										
Test Set	Set 1	Set 2	Set 3	Set 4	Set 5	Set 6	Set 7	Sets 1,5	Sets 1,6	Sets 5,6	Sets 1,5,6
Set 1	96.30	95.70	95.20	94.10	89.40	92.90	94.40	96.15	95.51	92.83	92.41
Set 2	97.90	98.80	97.90	97.90	95.50	96.20	97.70	96.91	97.08	95.32	98.17
Set 3	92.80	91.70	95.30	92.15	83.10	89.50	90.80	96.28	95.77	93.05	93.43
Set 4	96.80	97.50	96.70	97.80	93.02	95.90	95.50	98.13	98.22	97.81	98.05
Set 5	92.90	93.30	92.70	92.80	91.98	92.29	90.40	94.8	91.61	92.87	94.08
Set 6	76.70	80.20	78.00	76.40	65.38	89.08	72.28	67.32	85.33	88.47	86.32
Set 7	98.30	99.20	99.50	99.60	98.00	99.30	100	99.7	100	100	100

Set 1 includes natural objects and set 4 includes artificial objects, see Table 2 for more results.

After this, we set up the second layer to classify the MNIST handwritten digits by using 60,000 data for training and 10,000 data for test. As a result, it could classify 10 groups (0–9) of the handwritten digit images of the test set with 93.30% accuracy. It is interesting to know that by using the MNIST exclusively for training the features, the rate of correct accuracy was 91.98%. The small difference between the results shows that both sets had the same statistical proportions, giving rise to quite optimal features used for digit classification.

In comparison to this, the result of the handwritten digit recognition by the LDA classifier implemented in the Matlab software package was only 87.6%. The best result for handwritten digit classification reported in literature is 99.77% for the training error and was obtained using a special six layer nonlinear neural network, each layer stacked on top of another one (convolutional neural network).[3] Consider that this result was not obtained by universal features and their test set results should be worse our results are very good.

The state-of-the-art result for ALOI dataset classification is 99.8%.[34] If we use the same dataset, both for feature learning and classification, the result was about 1% better than using a different set for feature learning. This means that, by using universal feature extraction, we loose almost no accuracy.

In Table 2, you see more results of this test. This table displays the rate of accuracy (in percent) with multiple training and test sets for feature extraction and classification. In the last four columns, we used feature weights trained by the union of multiple data sets. We can see that, if the training set material is sufficiently rich, adding other sets to this set does not have much effect on the rate of accuracy, though the result is a little bit better. In Fig. 14, we can see the effect of using multiple data sets for training. It shows that, by using more data sets, the results are about the same or also improve. In the case of handwritten digits, the result of using multiple sets of different data for feature extraction is much better than using only handwritten digits both for feature extraction and classification. The digits are graphically very simple, using statistically more diversified images in the training will lead to more complex features and improve the results. The low accuracy of class 6 by

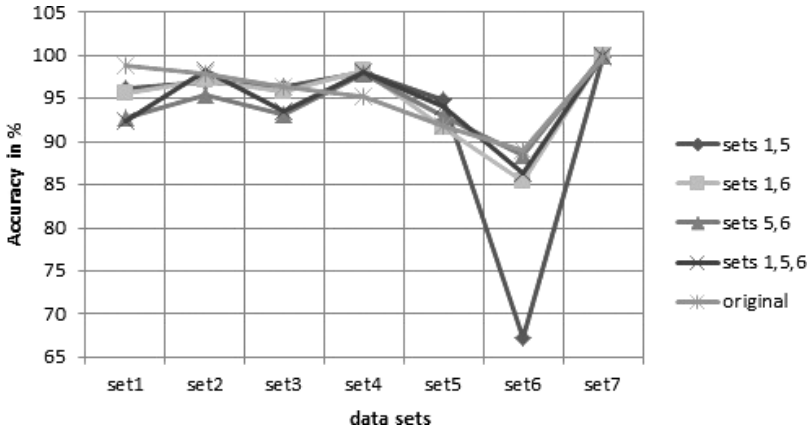

Fig. 14. Effect of using multiple data sets as classifier.

a system trained only with the union of sets 1 and 5, may be due to the fact that the statistical diversity within set 6 is extraordinarily high. So, the class boundaries can be hardly found to the overlapping material of sets 1 and 5. Here, higher level form features are demanded for recognition.

4.4. *Does the network learn also scale invariant features?*

In this test, we used the data sets with 40,000 objects with three different scales (1, 2 and 2.5) and also with different illumination and view angles to train the system (see Fig. 9). After training, we used the features to classify the data sets which include multiple scales, illuminations and view angles, and we got the accuracy of 82.91% and 82.11% for set 8 and 9, respectively. These results were 82.20% and 81.35% for sets 10 and 11, respectively. The smaller accuracy reflects the fact that, we use a static system which does not adapt to possible changes of input. There are a lot of systems which try to cope with this problem, but this is outside the focus of this paper. For more details, see Table 3.

4.5. *Changing the size of the receptive field*

Changing the size of the receptive field from 9×9 to 19×19 and the receptive field share to an overlap of 6 results in an accuracy of 86.20%, 88.61%, 86.35% and 92.13%, respectively for sets 1, 2, 3 and 4. It means that, by increasing the size of

Table 3. The accuracy rates for multi-scale objects.

Set	RF Size	RF Share	Feature Number	Accuracy %
Set 8	9	3	16	82.91
Set 9	9	3	16	82.11
Set 10	9	3	16	82.20
Set 11	9	3	16	81.35

Table 4. Effect of changing the size of receptive fields.

Set	RF Size	RF Share	Accuracy (%)	RF Size	RF Share	Accuracy (%)	RF Size	RF Share	Accuracy (%)
Set 1	7	2	**86.48**	9	3	**90.93**	19	6	**86.20**
Set 2	7	2	**90.67**	9	3	**91.17**	19	6	**88.61**
Set 3	7	2	**89.62**	9	3	**90.56**	19	6	**86.35**
Set 4	7	2	**95.96**	9	3	**95.95**	19	6	**92.13**

receptive field, the rate of accuracy is reduced a little. By decreasing the RF size to 7, we see that the rate of accuracy is also reduced a little. Therefore, the size should neither be too small nor too big. In all of these tests, set 1 has been selected for feature training. For more details, see Table 4.

4.6. *What kind of feature does the network learn?*

It is interesting to visualize the feature extractor filters that the network learned after training. In other words, we want to know if our filter or feature extractor looks like one of those filters found in literature, e.g. a Gabor filter, a differential of Gaussian (DOG) or some rotated bars. For this, we plot the weights of a receptive fields of some features as images in Fig. 15. They can be interpreted as filters. This features

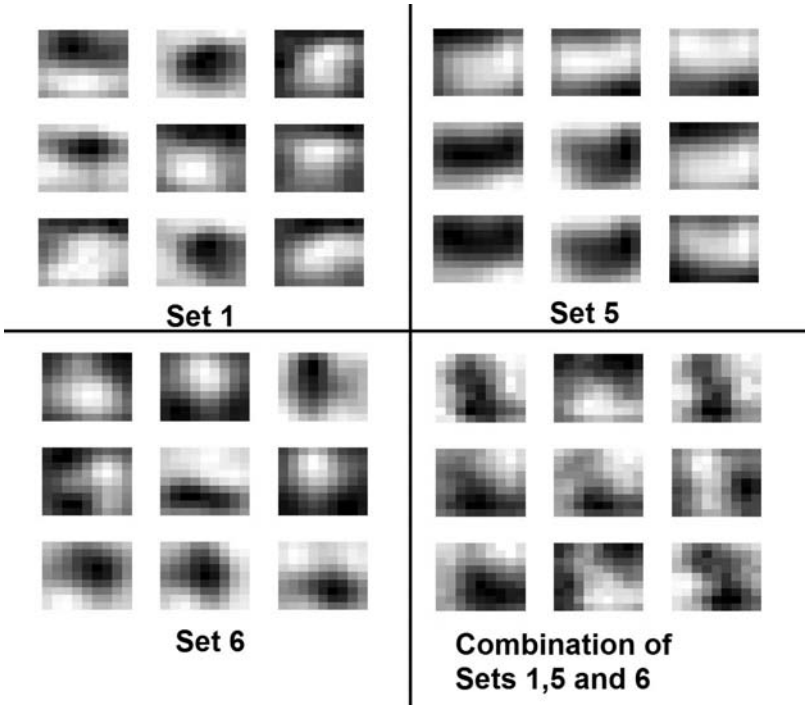

Fig. 15. Receptive field weights learned by different training sets.

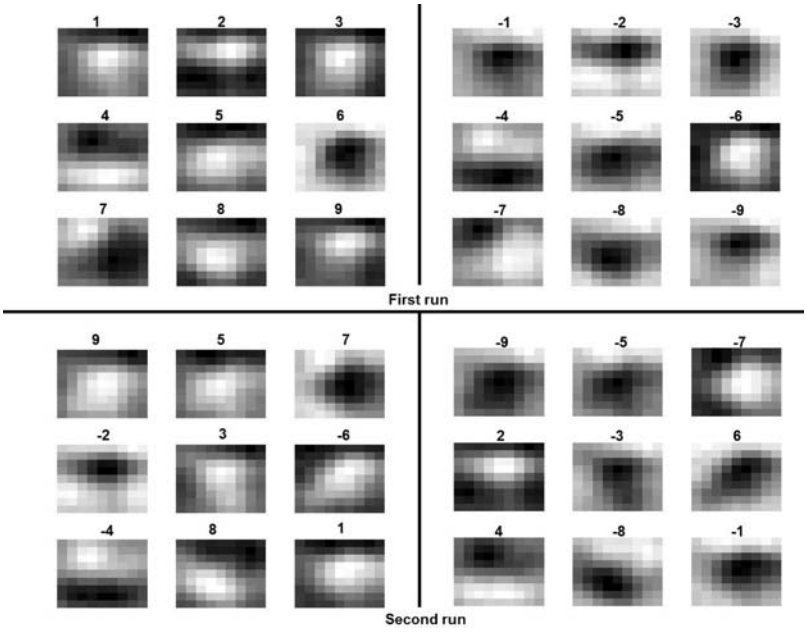

Fig. 16. Receptive field weights resulting from different runs but the same training pattern set using random starts. "-" sign refers to the negative of the weights.

are trained in parallel, so we learned all features simultaneously. The number of features was nine for these sets. Figure 15 shows a set of quite complex filters. Here, we used the neighbors influence as specified in formula 37. The resulting weights and therefore the features are unique in any run of the network from random initial

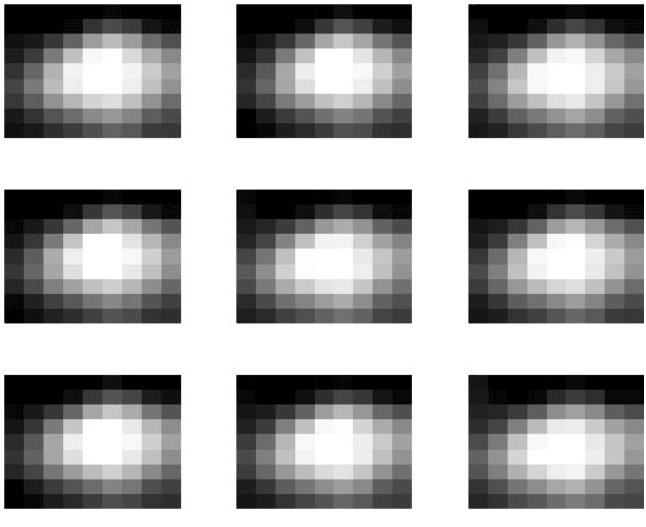

Fig. 17. Receptive field weights resulting from very small random initialization weights (0.00001).

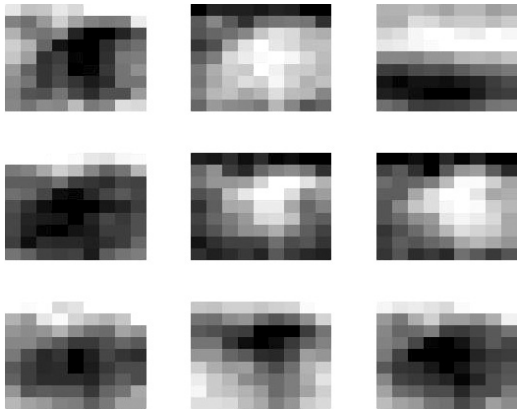

Fig. 18. Receptive field weights when we drop the neighborhood influence.

weights, but the numberings are different, see Fig. 16. This uniquely depends on starting conditions of weights and for this result it should be in $[-0.01, +0.01]$. By increasing this boundary to around ± 0.02, the result will not be unique and by decreasing this boundary to around $\pm 10^{-5}$, the result is unique but most of them looks like each others (see Fig. 17).

It may be interesting to see the weights for the case where we drop the neighborhood influence, in Fig. 18 you can see the result when we drop the neighborhood influence. It is shown that in this case, the weights are less smooth than before. Consider that by influencing the weights by their neighbors, the performance does not change significantly. This influence is biologically plausible and produces results which may be interpreted as filters.

4.7. *Changing the number of features (hidden units)*

In this test, all configuration and initialization was done as in Sec. 4.3 except that we increased the number of features to 25 and decreased them from 16 to 9 and 7. It is clear that a small number of hidden units (features) generalizes better than a bigger number, but might not be precise enough. On the other hand, a bigger number of features might be precise in training, but might fail to generalize due to over-fitting the training data. The results of the experiments are shown in Table 5. In all tests, we used set 1 for training the features.

Table 5. The effect of changing feature numbers.

Set	Features	Accuracy (%)	Features	Accuracy (%)	Features	Accuracy (%)	Features	Accuracy (%)
Set 1	25	**84.48**	16	**90.18**	9	**92.07**	7	**86.87**
Set 2	25	**91.11**	16	**90.08**	9	**91.12**	7	**91.65**
Set 3	25	**90.60**	16	**96.77**	9	**95.16**	7	**90.15**
Set 4	25	**96.43**	16	**93.71**	9	**93.32**	7	**95.00**

By decreasing the features from 9 to 7, the rate of accuracy is also reduced a little bit (see Table 5), whereas increasing the number of features takes much more computing time, but did not increase the performance significantly. Therefore, in our case, nine features present the best compromise (with average accuracy of 92.91% for four data sets) between generalization error and over-fitting error.

4.8. *Classification with random features*

It is interesting to compare the network performance with a network of random features. In this case, we do not learn any features but initialize the feature layer of network by random values.

Now to test the network with random features, we set the feature layer by random values of $[-0.5, +0.5]$ with uniform distribution and then tried to train the classification layer. The accuracy rate obtained for set 2 and set 4 was 58% and 63%, respectively. These amounts show that using random features cannot provide good result for our tasks.

4.9. *The extracted features and Gabor filters*

It is well known[17,26] that the simple cell (V1) response in the visual cortex of mammalian brain can be modeled by a Gabor filter. Now, how do the extracted features compare to Gabor filters? For this comparison, we picked up some of the extracted features and measured the Euclidean distance among them and Gabor filters with different scale and orientation. It means that for each extracted feature, we tried to find the best match Gabor filter which has the minimum distance with our selected feature as a pair of our features. In Fig. 19, the Euclidean distance among Gabor and our extracted filter is displayed between nine extracted features from sets 1, 2 and 3 and the appropriate Gabor filter. In this figure, the distance values are displayed between zero (minimum distance) and one (maximum distance). We see that the extracted feature are similar to Gabor filters with the average distance of 0.55. The reason that the extracted features are not very close to Gabor filters my be refer to some preprocessing algorithms like whitening. For instance in Ref. 5, the extracted features were more like to the Gabor filters after they applied Mahalanobis whitening on the training images.

4.10. *Shallow network features versus auto-encoder features*

In this section, we compare the features of the proposed method with the features formed by an auto-encoder (AE) for reconstructing the input. Are the best features for classification also those who are the best for reconstructing the input? To answer this question, we set the input layer and hidden units of an AE to be the same as our best configuration for classification. Then, we set the input layer of an AE to 81 units and the hidden units to 9. The AE is a simple network that tries to reproduce at its output what is presented at the input. The basic AE is, in fact, a simple neural

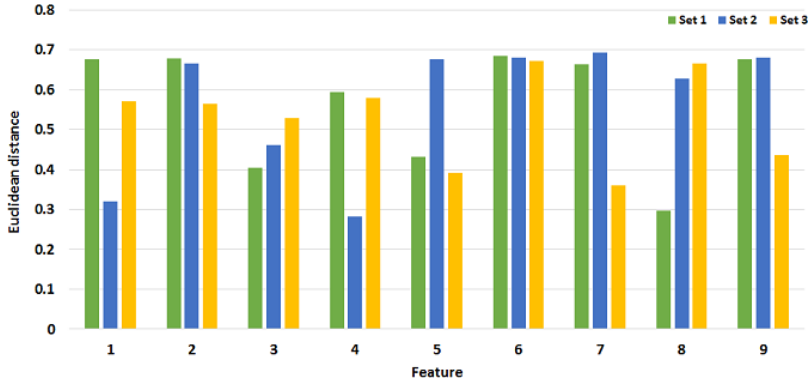

Fig. 19. Euclidean distance among nine extracted features from sets 1, set 2 and 3 with Gabor filters. The distance values are displayed between zero (minimum distance) and one (maximum distance).

network with one hidden layer and one output layer, subject to the number of output neurons is equal to the number of inputs. In Fig. 20, the structure of AE is shown with one hidden layer.

The relation between input neurons ($\mathbf{x}$) and output neurons ($\hat{\mathbf{x}}$) in a typical AE is as:

$$h(\mathbf{x}) = s(\mathbf{W_e}x + b) \tag{42}$$

and

$$\mathbf{x} = s(\mathbf{W_d}h(\mathbf{x}) + c), \tag{43}$$

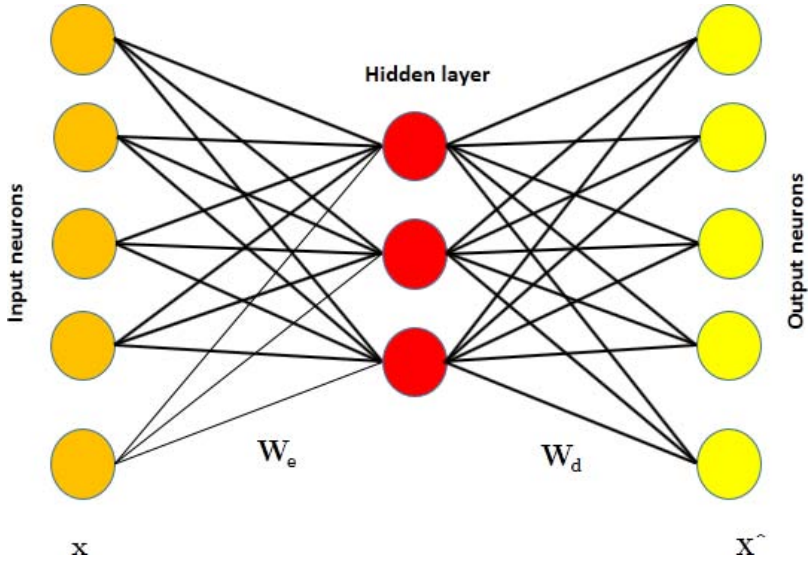

Fig. 20. A typical architecture of an AE with one hidden layer.

Table 6. Classification accuracies for various combinations of training and test sets using AE.

	Training Set			
Test Set	Set 1	Set 2	Set 3	Set 4
Set 1	85.39	85.19	83.27	84.83
Set 2	86.53	85.85	86.29	86.91
Set 3	92.89	93.45	93.31	94.38
Set 4	92.93	91.86	92.77	93.19

where s is a squashing function (e.g. tanh) and b and c are bias constants. To use the AE as feature extraction layer of the classifier, we have to train it with some unlabeled natural images. In this test, we used the same data sets which used for our proposed shallow network but here we used them as unlabeled data and extracted some patches from them randomly as a train sets for AE. As in any neural network, here also we have to define an objective function which is to be optimized. In general, there are several possible choices for the objective function e.g. mean squared error, cross entropy, etc. To have a good comparison between the result of classification by AE and our shallow network, we used cross entropy as objective function because our objective function in the shallow network was based on cross-entropy. Then, the objective function is defined by:

$$R = -\sum_k x_k \ln(\hat{x}_k) + (1 - x_k) \ln(1 - \hat{x}_k). \tag{44}$$

The gradient descent method is employed to minimize this objective function. Table 6 illustrates the accuracy rate of classification using the AE. To have a good comparison, we used tanh for squashing function in hidden layer and sigmoid for the output layer which we did for our shallow network. Comparing the result of two tables, it is obviously clear that the average classification rate for two methods are very close to each others ($88.93 \pm 0.60\%$ for shallow network and $89.31 \pm 0.43\%$ for AE). The interpretation of this may be that the AE tries to learn the features which are best for reconstruction while our network tries to learn the features which are better for classification, which is evidently not the same.

5. Conclusion

In this paper, we proposed a new method for universal feature extraction.

First, we used an information theory approach to design a proper risk function which leads to cross-entropy minimization. It is emphasized in some literatures that the cross-entropy risk function has significant, practical advantages over mean squared-error approaches.[14,21] We developed a feed forward neural network as basic structure to extract universal features.

Second, to reduce the number of parameter to learn, as constraint we used a weight sharing method for all receptive fields. In addition of reducing the number of learning parameters, it has the benefit that the shared weights makes all neurons detecting the same features, independent of their different positions in the input image. As draw-back, it should be noted that this decision is not covered by the original proofs[16] for the approximation properties of two-layer neural networks. Additionally, the labeling of the filter properties of the first layer as "features" is plausible, but arbitrary.

Nevertheless, the results show that those "universal features" are unique and can be successfully applied in very different image processing applications e.g. hand-written digit classification, recognition of natural or artificial objects which are placed in the natural or artificial background images and recognition of texture. The concept of "extreme learning" does not provide any good results here.

We used very different image sets for training and testing image features for classification purposes. Additionally, we changed several network parameters (e.g. network layer, number of hidden unit and size of receptive field) to get the best results. By these tests, we can give some answers to our questions posed in Sec. 3.1.

- What is the best size of a receptive field (patch)? The optimal RF size is 9×9.
- What is the optimum number of hidden units? The number of hidden units seems to be 16.

Although the universal features are a good start for really recognizing natural objects in images additional questions have be studied:

- The approach has shown the abilities of shallow networks—but what about deep networks[38,39]? Is there a performance increase possible?
- How can we make the system invariant to the position of objects in image so it could recognize objects not only in the center of background image, but also in any places of image?
- How can we make the system to adapt to different shadings and object sizes?
- How can the optimal size of the receptive fields be obtained automatically?

Here, more dynamical architectural approaches have to be developed.

References

1. S. Arivazhagan and L. Ganesan, Texture classification using wavelet transform, *Pattern Recogn. Lett.* **24**(9–10) (2003) 1513–1521.
2. B. Caputo, E. Hayman and P. Mallikarjuna, Class-specific material categorisation, In *Tenth IEEE Int. Conf. Computer Vision, 2005, ICCV 2005*, Vol. 2 (2005), pp. 1597–1604.
3. D. C. Ciresan, U. Meier and J. Schmidhuber, Transfer learning for latin and chinese characters with deep neural networks, In *The 2012 International Joint Conf. Neural Networks (IJCNN)*, Brisbane, Australia, June 10–15, (2012), pp. 1–6.

4. T. W. S. Chow and M. K. M. Rahman, A new image classification technique using tree-structured regional features, *Neurocomput.* **70**(4–6) (2007) 1040–1050.
5. A. Coates, H. Lee and A.Y. Ng, An analysis of single-layer networks in unsupervised feature learning, in *Proceedings of the Fourteenth International Conf. on Artificial Intelligence and Statistics*, eds. G. Gordon, D. Dunson and M. Dudk, Volume 15 of *JMLR Workshop and Conference Proceedings* (JMLR W&CP, 2011), pp. 215–223.
6. T. Cover and J. Thomas, *Elements of Information Theory* (Wiley & Sons, New York, 1991).
7. 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 *Advances in Neural Information Processing Systems*, Morgan Kaufmann (1990), pp. 396–404.
8. C. F. Tsai, Image mining by spectral features: A case study of scenery image classification, *Expert Syst. Appl.* **32** (2007) 135–142.
9. R. Duda, P. Hart and D. Stork, *Pattern Classification* (Wiley & Sons, New York, 2000).
10. W. Duch, R. Adamczak and N. Jankowski, Initialization and optimization of multilayered perceptrons, In *3rd Conf. Neural Networks and their Applications* (1997), pp. 105–110.
11. Fawcett and Tom, An introduction to roc analysis, *Pattern Recogn. Lett.* **27**(8) (2006) 861–874.
12. M. Fernandez-Redondo and C. Hernandez-Espinosa, Weight initialization methods for multilayer feedforward, *ESANN'2001 Proceedings —European Symposium on Artificial Neural Networks Bruges (Belgium), D-Facto Public., ISBN 2-930307-01-3*, April 25–27 (2001), pp. 105–110.
13. J. M. Geusebroek, G. J. Burghouts and A. W. M. Smeulders, The amsterdam library of object images, *Int. J. Computer Visi.* **61**(1) (2005) 103–112.
14. P. Golik, P. Doetsch and H. Ney, Cross-entropy vs. squared error training: A theoretical and experimental comparison, In *INTERSPEECH'13* (2013), pp. 1756–1760.
15. M. Haibach, Informationsgesteuerte, adaptive extraktion von merkmalen, Master's Thesis, Computer Science Dep., Goethe-Universit (2011).
16. K. Hornik, Approximation capabilities of multilayer feedforward networks, *Neural Netw.* **4** (1991) 251–257.
17. D. H. Hubel and T. N. Wiesel, Receptive fields of single neurons in the cat's striate cortex, *J. Phys.* **148** (1959) 574–591.
18. Md. M. Islam, D. Zhang and G. Lu, A geometric method to compute directionality features for texture images, In *ICME,conf/icmcs/2008*, IEEE, February 2008, pp. 1521–1524.
19. B. Karlik and A. V. Olgac, Performance analysis of various activation functions in generalized mlp architectures of neural networks, *Int. J. Artif. Intell. Expert Systems (IJAE)*, **1**(4) (2011) 111–122.
20. Y. K. Kim, Weight value initialization for improving training speed in the back-propagation network, *Proc. Int. Joint Conf. on Neu. Netw.* **3** (1991) 2396–2401.
21. M. Kline and L. Berardi, Revisiting squared-error and cross-entropy functions for training neural network classifiers, *Neur. Comput. Appl.* **14**(4) (2005) 310–318.
22. A. Krizhevsky, I. Sutskever and G. E. Hinton, Imagenet classification with deep convolutional neural networks, In *Advances in Neural Information Processing Systems (NIPS 2012)* (2012) p. 4.
23. S. Lazebnik, C. Schmid and J. Ponce, A sparse texture representation using local affine regions, *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, **27**(8) (2005) 1265–1278.

24. W. H. Leung and T. Chen, Trademark retrieval using contour-skeleton stroke classification, In *ICME (2),conf/icmcs/2002-2*, IEEE 2002, pp. 517–520.
25. S. Li and J. Shawe-Taylor, Comparison and fusion of multiresolution features for texture classification, *Pattern Recogn. Lett.* **26**(5) (2005) 633–638.
26. S. Marĉelja, Mathematical description of the responses of simple cortical cells, *J. Opt. Soc. Am.* **70**(11) (1980) 1297–1300.
27. T. Mitchell, *Machine Learning* (McGraw Hill, 1997).
28. C. E. Metz. Basic principles of roc analysis, *Semin. Nucl. Med.* **8**(4) (1978) 283–298.
29. L. J. Quackenbush, A review of techniques for extracting linear features from imagery. *Photogramm. Eng. Rem. Sens.* **70**(12) (2004) 1383–1392.
30. R. Raina, A. Battle, H. Lee, B. Packer and A. Y. Ng, Self-taught learning: Transfer learning from unlabeled data, In *Proceedings of the 24th International Conference on Machine Learning*, ACM (2007), pp. 759–766.
31. J. Schmidhuber, Deep learning in neural networks: An overview, *Neural Networks*, **61**(0) (2015) 85–117.
32. P. Sibi, S. A. Jones and P. Siddarth, Analyses of different activation functions using back propagation neural networks, *J. Theor. Appl. Inform. Tech.* **47**(3) (2013) 1264–1268.
33. T. K. Shih, J. Y. Huang and C. S. Wang, An intelligent content-based image retrieval system based on color, shape and spatial relations, In *National Science Council ROC(A)*, Vol. 25 (2001), pp. 232–243.
34. P. Smagghe, J.-L. Buessler and J.-P. Urban, Novelty detection in image recognition using irf neural networks properties, In *ESANN-2013*, 2013.
35. D. P. Tian, A review on image feature extraction and representation techniques, *Int. J. Multimed. Ubiquit. Enginee.* **8**(4) (2013) 385–396.
36. P. L. Stanchev, D. Green Jr. and B. Dimitrov, High level colour similarity retrieval, *Int. J. Inform. Theories Appl.* **10**(3) (2003) 363–369.
37. D. Tjondronegoro, Y. Liu, J. Zhang and S. Geve, A shape ontology framework for bird classification, 2007.
38. C.-F. Tsai and W.-C. Lin, A comparative study of global and local feature representations in image database categorization, in *NCM*, (2009), pp. 1563–1566.
39. N. C. Yang, W. H. Chang and C. M. Kuo, A fast mpeg-7 dominant colour extraction with new similarity measure for image retrieval, *J. Vis. Comm. Image Retrieval* **19** (2008) 92–105.

Mohammad Amiri was born in Sari, Iran, in 1978. He received his B.Sc. degree in Computer Engineering from the Sharif University of Technology, Tehran, Iran, in 1999, and his Masters degree in Computer System Architecture from the Isfahan University, Isfahan, Iran, in 2002. In 2013, he joined the Department of Computer science, Goethe University, Frankfurt, Germany as a PhD student. His current research interests include machine vision, machine learning, pattern recognition and neural network.

Rüdiger W. Brause studied Physics and Cybernetics at Tübingen in Germany where he received his diploma in Physics. His Ph.D. thesis treated the adaptive diagnosis of wafer-scale multiprocessor systems. After joining a fault-tolerant load balancing multiprocessor system project, he changed to the Goethe-University of Frankfurt where he headed a working group for adaptive information processing systems. Since 2005, he has been a Professor for Applied Informatics and the author of several books on operating systems and neural networks. His current research interests are concentrated on encoding and adaptive, content-oriented information extraction in images, medical and financial applications.