Revista de Sistemas de Informação da FSMA
n. 22 (2018) pp. 49-59

SISTEMAS DE INFORMAÇÃO
http://www.fsma.edu.br/si/sistemas.html

TRILHA PRINCIPAL

A Comprehensive Introduction to Convolutional Neural Networks: A Case Study for Character Recognition

Elivelto Ebermam, Federal University of Espírito Santo, Vitória - ES
Renato A. Krohling, Federal University of Espírito Santo, Vitória - ES

***Abstract*—Convolutional neural networks have been attracted great attention in the field of complex tasks, mainly in image recognition. They were specifically designed to handle images as inputs, as they act in local receptive fields performing a convolution process. However, understanding the working principle of convolutional neural networks may not be an easy task, especially for beginners in the area of computational intelligence. So, the aim of this work is to present in a didactic and intuitive way the convolutional neural networks. A case study involving alphabet character recognition is presented in order to illustrate the feasibility of the approach.**

***Index Terms*—Convolutional neural networks, character recognition.**

I. Introduction

ARTIFICIAL neural networks have been applied in several areas of knowledge, such as pattern recognition, character recognition, time series forecasting, among others [1]. Since they first appeared, they have been improved, adapting to several applications, one of which is image classification, for which the most widely used is the convolutional neural network (CNN).

The CNN [2] has a structure specially built to receive images as inputs. It can preserve the correlations among neighboring pixels because it acts on the local receptive fields, performing the convolution operations. This way, we can reduce the sensitivity to image translation, rotation and distortion [3]. Other types of neural networks cannot capture this kind of relationships, because they consider images as an unidimensional array.

Nevertheless, the convolutional neural networks are more complex than other neural networks architectures, what makes it more difficult for beginners in the area of computational intelligence to understand the way it works.

Even though there are several papers on convolutional neural networks in the literature, just a few of them aims at a comprehensive introduction. O'Shea and Nash [4] give a brief introduction to CNNs, discussing recent papers and techniques for their development. Wu [5] discusses CNNs mathematically in a more clear way. In Brazil, as far as the authors know, there are few introductory articles on the subject. Araújo et al. [6] describe the CNNs main concepts, besides presenting the model in a practical way.

The problem of character recognition is well known in literature. LeCun et al. [7] used CNNs for a dataset of digits known as MNIST. For the same base, Mohebi and Bagirov [8] used a modified self-organizing maps (SOM). Bai et al. [9] used a variation of CNN to recognize characters from different languages. Besides character recognition, CNN can be used in several applications, such as license plates automatic recognition. Another examples of applications include document digitalization, receipt images, check processing, medical services form processing and others [10].

The goal of this work is to present convolutional neural networks in a didactic and intuitive way. For that, we will make a brief introduction to artificial neuron and conventional neural network (called multi-layer perceptron) in section 2, because it is used as the final stage of a convolutional network. Section 3 discusses in details the elements of a convolucional neural network, focusing mainly on the convolution and pooling operations. In order to facilitate the understanding, we present a case study on alphabetic character recognition in section 4. Finally, we present the conclusions in section 5.

II. Artificial Neural Networks

ARTIFICIAL neural network is a parallel and distributed information processing system [11]. It presents similar characteristics to the biological neural network, such as parallel processing and the ability to learn [12].

Neural networks are made up of several simple processors called artificial neurons, each one of them producing a series of real valued activations [13]. The artificial neuron is the basic processing unit in an artificial neural network. Several models were proposed, but the most one used was created by McCulloch and Pitts [14].

An artificial neuron basically consist of input signals, weights, activation function and output signal, as shown in Figure 1. Each input x_i is multiplied by a weight w_i and the resulting values are summed. A bias value b is added and thus an activation signal u is generated (calculated by

the formula $u = \sum_i x_i \cdot w_i + b$). This signal is then passed through an activation function $f(u)$ resulting in the output y.

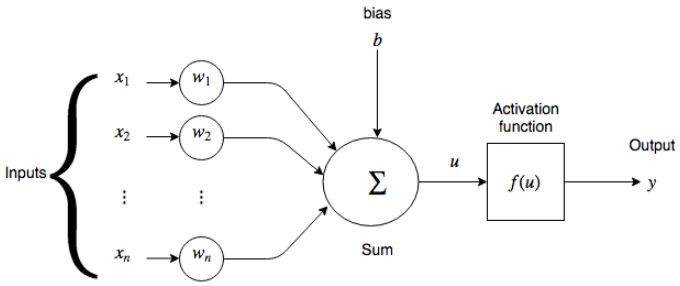

Fig. 1. Artificial neuron [15]

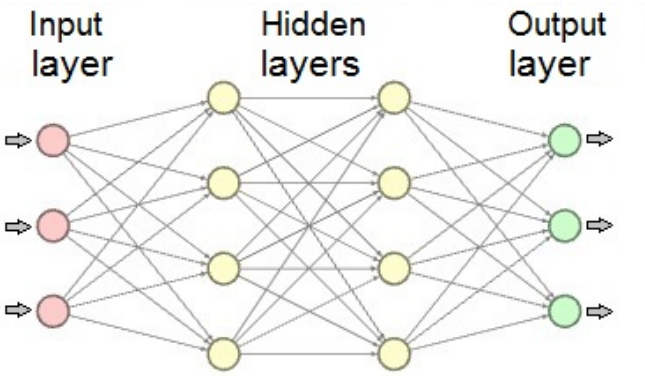

Fig. 2. Structure of a multi-layer artificial neural network

The most common activation functions are: logistic sigmoidal, hyperbolic tangent and rectified linear unit (ReLU). These functions are calculated by the equations 1, 2 and 3 respectively.

$$f(u) = \frac{1}{1 + e^{-u}} \tag{1}$$

$$f(u) = \frac{1 - e^{-u}}{1 + e^{-u}} \tag{2}$$

$$f(u) = max(0, u) \tag{3}$$

It is easier to understand the process through an example. Given a neuron with a logistic sigmoidal activation function, inputs $[x_1 = 0,\ x_2 = 0.5,\ x_3 = 1]$, weights $[w_1 = 0.1,\ w_2 = 0.2,\ w_3 = 0.7]$ and bias $[b = -0.8]$, one generates the output 0.5, according to the following calculations:

$$\begin{aligned}
u &= \sum_i x_i \cdot w_i + b \\
u &= x_1 \cdot w_1 + x_2 \cdot w_2 + x_3 \cdot w_3 + b \\
u &= 0 \cdot 0.1 + 0.5 \cdot 0.2 + 1 \cdot 0.7 - 0.8 \\
u &= 0 \\
y &= f(u) \\
y &= \frac{1}{1 + e^{-u}} \\
y &= \frac{1}{1 + e^{-0}} \\
y &= \frac{1}{1 + 1} \\
y &= 0.5
\end{aligned} \tag{4}$$

Several connected neurons form an artificial neural network, whereas the output y of a neuron is the input of other neurons in the next level. It is common that they are grouped into layers and their connection is unidirectional, with the input going "forward"(feedforward). The structure of a neural network is illustrated in Figure 2.

The first layer is called input layer, which only receives data and transmits them to the following layer. The intermediate layers are called hidden layers, being responsible for information processing. Finally, the output layer, whose goal is to present the network's response (output) to the problem at hand, which can be either a class identification or a continuous value.

III. Convolutional Neural Networks

A. Introduction

Convolutional neural networks are a kind of neural networks inspired by the animal visual cortex [16]. The initial layers are specialized in extracting features from data (specially from images), and the neurons are not totally connected to the next layer. The final layers are totally connected, as shown in Figure 2 and are responsible for interpreting the information extracted by the initial layers and offering a response. Convolutional neural networks are based on the work of Hubel and Wiesel [17], who studied the visual field of cats, identifying that visual cells respond to different types of stimuli.

In 1980, Fukushima introduced a neural network called Neocognitron. This network is self organized using unsupervised learning (which does not need a tutor) and acquires the ability to recognize patterns of stimuli based on geometric similarity [18].

This network structure is made up of multiple layers with hierarchy levels, being made of layers of simple, complex and hyper-complex cells. These cells respond to specific types of stimuli from receptive fields. At each level, the complexity of the features extracted from the stimuli increases, in a similar way to the visual nervous system.

In 1986, there was a huge breakthrough in artificial neural networks, with the work of Rumelhart et al. [19], who developed a supervised learning algorithm known as *backpropagation*, which provide the networks the ability to solve nonlinear problems.

In 1989, LeCun [2] used hierarchically structured neural networks with invariance to translation detectors called "*Multilayer Constrained Networks*". In those networks, the connections were made locally but using shared weights and only the final layers were fully connected. The training was carried out in a supervised way using a variation of the backpropagation learning algorithm.

In 1994, LeCun et al. [3] referred to the networks previously referenced as “Multilayer Convolutional Neural Network” or simply “Convolutional Neural Network”. In 1994, the term “Convolutional Neural Networks” appeared in the title of a paper by LeCun et al. [20] and in 1995, LeCun and Bengio [21] published a paper dedicated to convolutional neural networks and their applications. In 1998, LeCun et al. [7] described a model of convolutional neural network with a large number of layers (seven in total) called LeNet-5. In 2012, Krizhevsky et al. [22] developed a model of CNN called AlexNet, made of 8 layers. They used the model to perform classification within a large database, which store millions of images, called ImageNet. In order to train the network, they used graphic processing units (GPUs).

Ever since, with the increase of computational power, the number of layers of Convolutional Neural Networks increased, e.g., dozens or even hundreds of layers.

B. Architecture of a convolutional neural network

Convolutional neural networks are feedforward neural networks designed to minimize sensitivity to the input image translation, rotation and distortion [3]. They are built of local connections, shared weights, pooling and the use of several layers [23].

The network is organized into layers with different purposes. The convolutional layer is responsible for extracting image features, the pooling layer is responsible for performing a sub-sampling of the image and the fully connected layers are responsible for the interpretation of the extracted features.

C. Convolutional layer

In the convolutional layer, the units are organized into “feature maps”, in which each unit is connected to a part of the previous layer by a set of weights called ”filters” [23]. The processing units (or neurons) use a technique called ”shared weights” which consists on several connections being defined by the same parameter (weight) [2]. This technique decreases the number of parameters of network. In addition, this type of connection organization simulates the convolution operation, which seeks to extract some image features, such as lines and contours.

In Figure 3 we illustrate the technique of weight sharing. The goal of this example is to show how the connections between the inputs and the neurons in the convolutional layer is made (the output of those neurons form the feature maps). Each neuron, in grey, connects to two inputs through a weight filter $[w_1, w_2]$. In a sequential way, we can imagine at first glance that the two weights are connected to the two first inputs and generate the first unit of the feature map. At second glance, the weights slide down and connect to the second and third input, forming the second unit of the feature map. Further on in this paper, we will show a case of a 2D input and a 2D filter.

The filters can also be organized in two or three dimensions. At each moment, they are connected to a specific

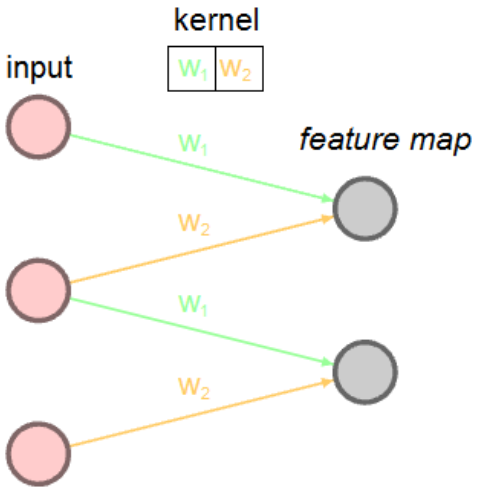

Fig. 3. Neuron weight sharing in the convolutional layer

part of a matrix called local receptive field. In this case, filters can be understood as small squares or cuboids that slide within a matrix. They have a specific size, such as 5x5 or 5x5x3 (for inputs in three dimensions) and a movement step called “stride”.

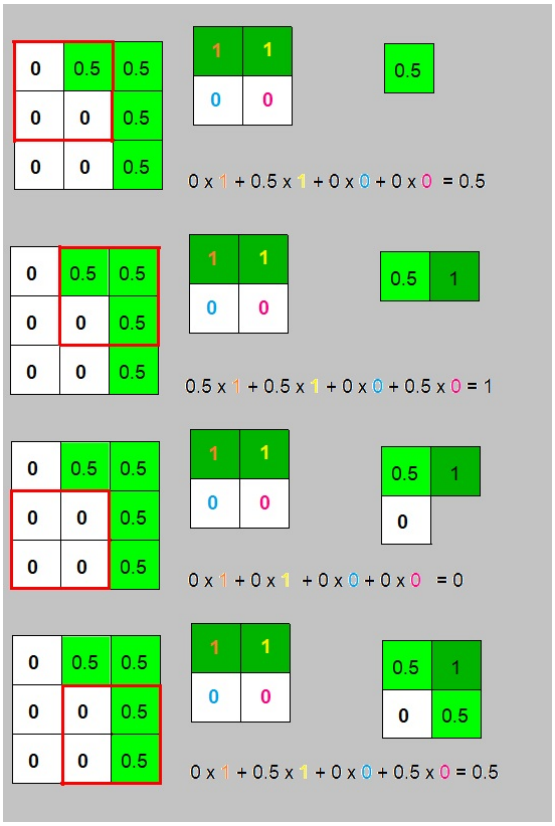

Fig. 4. Example of convolution

In Figure 4 we have an input image of size 3x3 with values between 0 (zero) and 1 (one), where 0 correspond to the lighter pixels and 1 to the darker ones. The filter used has size 2x2, with weights $[1, 1, 0, 0]$ and acts on the

local receptive fields (square with red borders) of the image of the same size (2x2). Each filter weight is multiplied by the value of the corresponding pixel in the receptive field. These values are summed and add to the bias term, resulting in a value for the corresponding unit of the feature map. In this example, the bias is not shown in the Figure 4 and, hence, its value was considered as equal to zero. The size of the stride used is one, which means that at each moment, the local receptive field is dislocated one pixel to the right. When we get to the end of a line (like in the second moment of the image), we shift one pixel down and go back to the beginning of the line.

As shown in Figure 4, the application of a filter (2x2) in an image (3x3) results in a feature map with size 2x2. The size of the feature map is calculated according to the following equation::

$$T_m = \frac{M - k}{s} + 1 \tag{5}$$

where T_m is the size of the feature map, M is the size of the image and k is the size of the filter kernel and s is the stride size. The variables T_m, M and k represent the size in one of the dimensions (horizontal or vertical), but it is more frequent to use square images and filters, which have equal lengths and heights. Hence, the variables would represent the sizes in both dimensions.

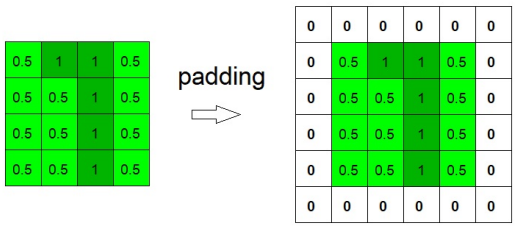

Fig. 5. Illustration of an example of zero-padding

In order to achieve a specific size for a feature map, sometimes it is necessary to change the size of the input image. This is done adding a border called padding or zero-padding, which consists in inserting zeros around the image, as illustrated by Figure 5. This way, the resulting size of the feature maps includes the width of the border p that is calculated according to the following formula:

$$T_m = \frac{M + 2p - k}{s} + 1 \tag{6}$$

After generating the values of the feature map, it is necessary to make them go through an activation function, This is done so that they become able to solve nonlinear problems (in this case, the activation function must also be nonlinear). The activation function most used with convolutional neural networks is the ReLU.

Algorithm 1 summarizes the process that is performed by the convolutional layer.

In the pseudocode, $x_{i,j}$ is the pixel value at position (i, j) of image x, M is the image height and N is the image width. The variable $u_{i,j}$ stores the value of position (i, j) of the feature map and $y_{i,j}$ is the output after going through the activation function. The variable $w_{c,d}$ indicates the filter weight at position (c, d).

Algorithm 1 Convolution

Initialize the values of the weights w with small values
For $i = 1$ to $(M - k)/s + 1$ **Do**
  For $j = 1$ to $(N - k)/s + 1$ **Do**
    $u_{i,j} = \sum_{c=1}^{k} \sum_{d=1}^{k} x_{i \cdot s-1+c, j \cdot s-1+d} \cdot w_{c,d} + b$
    $y_{i,j} = f(u_{i,j})$
  End-for
End-for

D. Pooling layer

The goal of the pooling layer is to reach a certain level of invariance towards rotation, reducing the resolution of the feature maps [24]. It acts similarly to the convolutional layer, but the stride has the same size as the filter (in the case of the previous layer, the stride was equal to 1). This makes the receptive fields to be totally different and reduces the feature maps by 75%. The most common types of pooling are max pooling and average pooling. In max pooling, we select the highest value of the receptive field and in average pooling, we calculate the average of the values.

In Figure 6, the pooling layer receives as input an image in shades of green and perform the max pooling operation on it. As a result, the output is an image with half of the original height and width.

The *pooling* operation performed after the convolutional layer is described in the algorithm 2 (considering $s = k$).

Algorithm 2 pooling

For $i = 1$ up to $(M - k)/s + 1$ **Do**
  For $j = 1$ up to $(N - k)/s + 1$ **Do**
    If *max pooling* **Then**
      $y_{i,j} = max(x_{(i-1) \cdot k+1, (i-1) \cdot k+1}, \ldots, x_{i \cdot k, j \cdot k})$
    End-If
    If *average pooling* **Then**
      $y_{i,j} = (\sum_{c=1}^{k} \sum_{d=1}^{k} x_{i \cdot s-1+c, j \cdot s-1+d})/(k^2)$
    End-If
  End-For
End-For

E. Fully connected layer

The input passes through the convolutional and pooling layers, which identify the features, from the simplest to the most complex, until they enter the fully connected layers. Usually, the previous layer is a pooling one, in which the feature maps have more than one dimension. Hence, they are redefined for a single dimension (as a vector), so that they can be connected to the final part of the network. The fully connected layers work as a multilayer feedforward neural network and are responsible for the interpretation of the features extracted by the initial layers.

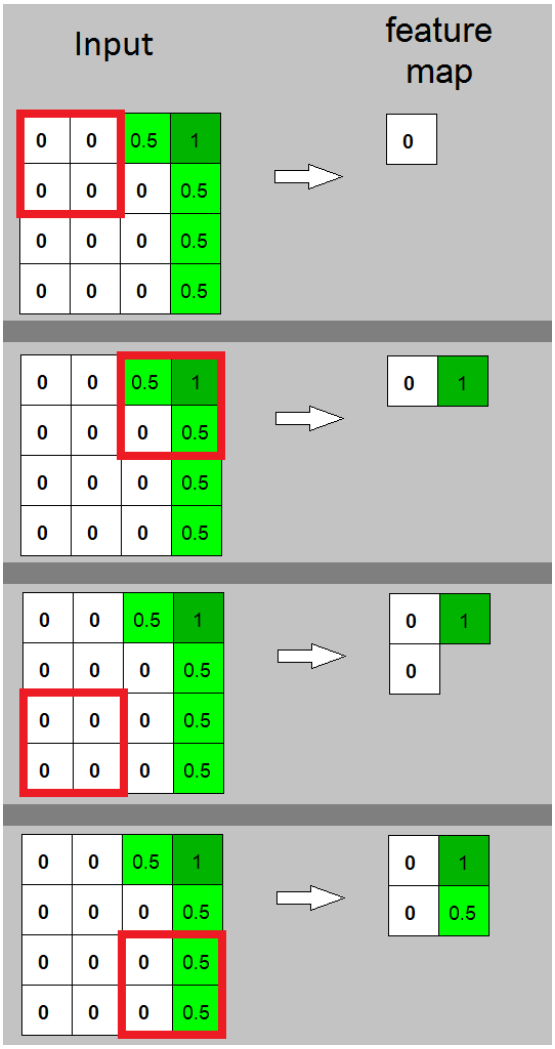

Fig. 6. Illustration of a max pooling operation

The pseudo-code for the fully connected layer processing is shown in the algorithm 3.

Algorithm 3 Fully connected layer

1: **If** previous layer is either a pooling or convolutional one **Then**
2: resize(x)
3: **End-If**
4: **For** each neuron j of the fully connected layer **Do**
5: **For** each input i from the previous layer **Do**
6: $u_j = \sum_{i=1}^{n} x_i \cdot w_{i,j} + b_j$
7: $y_j = f(u_j)$
8: **End-For**
9: **End-For**

F. Softmax

The final layer is responsible for presenting the results. The number of neurons is defined by the number of classes of the problem. Each neuron presents the output as a probability in the range $[0, 1]$)and the response is the neuron with the highest output. This is performed with the softmax function, described by the following equation:

$$y_j = \frac{e^{y(j)}}{\sum_{i=1}^{n} e^{y(i)}} \tag{7}$$

where the output y of neuron j is the output value divided by the sum of all the other outputs from the neurons in this layer.

For instance, consider that the neural network output before the application of the softmax function are: $[y_1 = 1, y_2 = 1.5, y_3 = 2]$. Hence, the outputs from the network will be calculated according to the following equations:

$$\begin{aligned}
y_j &= \frac{e^{y(j)}}{\sum_{i=1}^{n} e^{y(i)}} \\
y_j &= \frac{e^{y_j}}{e^{y_1} + e^{y_2} + e^{y_3}} \\
y_j &= \frac{e^{y_j}}{e^{1} + e^{1.5} + e^{2}} \\
y_j &= \frac{e^{y_j}}{2.72 + 4.48 + 7.39} \\
y_j &= \frac{e^{y_j}}{14.59} \\
y_1 &= \frac{e^{y_1}}{14.59} = \frac{e^{1}}{14.59} = \frac{2.72}{14.59} = 0.186 \\
y_2 &= \frac{e^{y_2}}{14.59} = \frac{e^{1.5}}{14.59} = \frac{4.48}{14.59} = 0.307 \\
y_3 &= \frac{e^{y_3}}{14.59} = \frac{e^{2}}{14.59} = \frac{7.39}{14.59} = 0.507
\end{aligned} \tag{8}$$

So, we complete the construction of the convolutional neural network, which is illustrated in Figure.7.

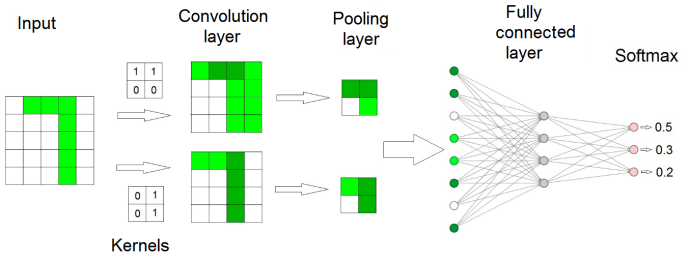

Fig. 7. Convolutional neural network

G. Learning Algorithm

Usually, learning is performed using the error back-propagation method, which uses the gradient descent to update the weights [24]. In order to calculate the error, it is necessary to define a specific cost function, such as the Euclidean distance between the network response and the expected value. Nevertheless, for classification, when there are more than two classes, it is used the cross entropy function described by the following equation:

$$C = -\sum_{i=1} y_i log(\hat{y}_i) \tag{9}$$

where C is the cost, y_i is the neuron output and $\hat{y}_i$ is the desired output.

The training process consists in minimizing the cost function by changing the weights. The update is performed according to the following equation:

$$\Delta w_i^t = \eta \frac{\partial C}{\partial w_i^t} \tag{10}$$

$$w_i^{t+1} = w_i^t - \Delta w_i^t \tag{11}$$

where w_i^t is the weight i of the network at the current time (t), η is the learning rate and Δw_i^t is the variation of weight i at the current time.

In order to smooth the gradient descent, we can use part of its previous movement by incorporating the momentum term, according to the following equation:

$$w_i^{t+1} = w_i^t - \Delta w_i^t + \alpha \Delta w_i^{t-1} \tag{12}$$

where α is the momentum and its value is an exponential decaying average of the previous weight variations.

In addition, the gradient descent can be described using a stochastic gradient descent (SGD). This method estimates the gradients based on single examples selected randomly [24]. In fact, the gradients are calculated based on a set of samples called mini-batch [24]. Using batches requires less weights updates than the incremental mode (one example at a time), and therefore, reduces training time (which can be high if there are many images).

At each iteration we apply the weight updating process, which is done until we complete the whole set of samples. The process of adjusting the weights for all training samples is called an epoch.

Some other algorithms based on gradient descent are: RMSProp, Adagrad, Adadelta, Adam, among many others.

H. Dropout

Sometimes, the network may learn too well the images in its training set and at the same time, be unable to have a good performance with other images unseen during the training. In order to avoid this overfitting in its learning, we use the dropout technique. It consists in withdrawing randomly some units from the neural network [25], as can bee seen in Figure 8. We define a probability p, that defines whether each unit may propagate its signal throughout the network.

I. Development Platforms

There are several tools which include the implementation of several mechanisms that help build convolutional neural networks, such as convolution, pooling, activation functions and learning algorithms.

TensorFlow [26] is a library for large scale machine learning developed by researchers from Google. It is normally used with the Python programming language through an API. It also supports C++ and Java.

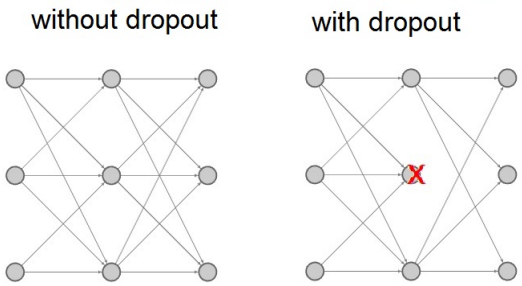

Fig. 8. Dropout. In the left, we see a network without dropout, with all its neurons active. In the right, one neuron in the second layer is randomly chosen to be "turned off", characterizing the use of dropout.

Keras [27] is a high level API for neural networks construction that works over other frameworks such as TensorFlow, CNTK and Theano.

Caffe [28] is an open source framework developped by the *Berkeley Vision and Learning Center* (BVLC). Its code was written in C++, using CUDA for graphic board computation, and it also provides libraries for Python/Numpy and MATLAB.

Torch [29] is a scientific computation framework which supports several machine learning algorithms. It is used through a script language called LuaJIT.

These tools make the process of building a neural network simpler and faster. Besides, by using them, the user can improve the performance of the neural network training using graphic processing units (GPUs) with no additional code.

It this paper, we used the Tensorflow framework, because it supports the Python programming language and is quite intuitive. In addition, it is very flexible, i.e, it allows the users to create and modify several structures within the neural network.

IV. Extreme Learning Machine

Extreme Learning Machine (ELM) is a learning algorithm for artificial neural networks with a single hidden layer [32]. The weights between the initial and the hidden layers are initialized randomly. The network training occurs in the weight matrix between the hidden and the output layer, which is calculated analytically. Since it does not require an iterative process, generally the network training is performed very quickly.

The first step of the ELM algorithm consists in defining the matrix H, which is the resulting matrix from the hidden layer neurons output for each of the samples of the input set, according to the following formula:

$$H = \begin{bmatrix} f(x_1 w_1 + b_i) & \dots & f(x_1 w_n + b_m) \\ \vdots & \ddots & \vdots \\ f(x_m w_1 + b_i) & \dots & f(x_m w_n + b_n) \end{bmatrix} \tag{13}$$

where the lines represent the samples and the columns represent the neurons in the hidden layer, m is the number

of input samples and n is the number of neurons in the hidden layer. In this case, x_i is the atribute vector and w_i is a weight vector.

The second step consists in calculating $\hat{\beta}$, the weight matrix between the hidden layer and the output layer. It is calculated as the solution of the least squared error (LSE) minimization problem for the linear system $H\hat{\beta} = T$, where $\hat{\beta} = H^{\dagger}T$, and $H^{\dagger}$ is the generalized Moore-Penrose inverse of the matrix H and T is the desired output vector for the input samples [32], [33].

Consider the problem of the logic function AND, in which there are two types of input: 1 (presence of an electrical current) and 0 (no electrical current). The logic gate receives two inputs and returns: 0&0 = 0, 0&1 = 0, 1&0 = 0 and 1&1 = 1. Hence, X and T are defined as follows:

$$X = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}, \quad T = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{14}$$

Consider also that the ELM has two neurons in the hidden layer with a logistic sigmoidal activation function and that the weights W and the bias B were generated randomly as follows:

$$W = \begin{bmatrix} 0.2 & 0.5 \\ 0.6 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} -0.5 & 0.5 \end{bmatrix} \tag{15}$$

So, $\hat{\beta}$ is calculated as follows:

$$\hat{\beta} = H^{\dagger}T$$

$$\hat{\beta} = (f(XW + B))^{\dagger}T$$

$$\hat{\beta} = \begin{bmatrix} f(x_{11}\dot{w}_{11} + x_{12}\dot{w}_{21} + b_1) & f(x_{11}\dot{w}_{21} + x_{12}\dot{w}_{22} + b_2) \\ f(x_{21}\dot{w}_{11} + x_{22}\dot{w}_{21} + b_1) & f(x_{21}\dot{w}_{21} + x_{22}\dot{w}_{22} + b_2) \\ f(x_{31}\dot{w}_{11} + x_{32}\dot{w}_{21} + b_1) & f(x_{31}\dot{w}_{21} + x_{32}\dot{w}_{22} + b_2) \\ f(x_{41}\dot{w}_{11} + x_{42}\dot{w}_{21} + b_1) & f(x_{41}\dot{w}_{21} + x_{42}\dot{w}_{22} + b_2) \end{bmatrix}^{\dagger} T$$

$$\hat{\beta} = \begin{bmatrix} f(0 \cdot 0.2 + 0 \cdot 0.6 + (-0.5)) & f(0 \cdot 0.5 + 0 \cdot 1 + 0.5) \\ f(0 \cdot 0.2 + 1 \cdot 0.6 + (-0.5)) & f(0 \cdot 0.5 + 1 \cdot 1 + 0.5) \\ f(1 \cdot 0.2 + 0 \cdot 0.6 + (-0.5)) & f(1 \cdot 0.5 + 0 \cdot 1 + 0.5) \\ f(1 \cdot 0.2 + 1 \cdot 0.6 + (-0.5)) & f(1 \cdot 0.5 + 1 \cdot 1 + 0.5) \end{bmatrix}^{\dagger} T$$

$$\hat{\beta} = \begin{bmatrix} f(-0.5) & f(0.5) \\ f(0.1) & f(1.5) \\ f(-0.3) & f(1) \\ f(0.3) & f(2) \end{bmatrix}^{\dagger} T$$

$$\hat{\beta} = \begin{bmatrix} \frac{1}{1+e^{-(-0.5)}} & \frac{1}{1+e^{-0.5}} \\ \frac{1}{1+e^{-0.1}} & \frac{1}{1+e^{-1.5}} \\ \frac{1}{1+e^{-(-0.3)}} & \frac{1}{1+e^{-1}} \\ \frac{1}{1+e^{-0.3}} & \frac{1}{1+e^{-2}} \end{bmatrix}^{\dagger} T$$

$$\hat{\beta} = \begin{bmatrix} 0.38 & 0.62 \\ 0.53 & 0.82 \\ 0.43 & 0.73 \\ 0.57 & 0.88 \end{bmatrix}^{\dagger} T$$

$$\hat{\beta} = \begin{bmatrix} -6.4520 & 6.9861 & -17.0639 & 12.2380 \\ 4.3023 & -4.0280 & 10.9916 & -7.2892 \end{bmatrix} T$$

$$\hat{\beta} = \begin{bmatrix} -6.4520 & 6.9861 & -17.0639 & 12.2380 \\ 4.3023 & -4.0280 & 10.9916 & -7.2892 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\hat{\beta} = \begin{bmatrix} 12.2380 \\ -7.2892 \end{bmatrix} \tag{16}$$

The calculation of the generalized inverse is a little more complex, but most programming languages have libraries to perform this calculation. Having the matrix $\hat{\beta}$, the output y of the ELM for a test sample Z is given as follows:

$$y = f(ZW + B) * \hat{\beta} \tag{17}$$

V. Case study

Even though MNIST is a standard digit recognition database widely used in the literature [30], in this work we will consider the problem of recognizing alphabetic characters from several sources to illustrate the application of CNN. It was chosen due to the fact that the latter problem is very similar to the former but the network needs to classify among 26 letters instead of only 10 digits, making the task a little bit more complex.

A. Database

The database 'Alphabet' was developed by students at the Labcin research lab (Nature Inspired Engineering and Computing Lab) from the graduate program in Computer Science. This database contains a total of 11,960 images of alphabetic characters in gray scale, both lowercase and capital letters, divided into 26 classes (each corresponding to a letter). Figure 9 shows some examples from this database.

Fig. 9. Samples from the Alphabet database.

The database was divided into three parts: 70% for training, 15% for validation and 15% for testing. The value of each pixel is in the interval [0,255], but was normalized in the [0,1] range.

B. Setting of the convolutional neural network

A convolutional neural network used[1] is presented in Figure 10. It receives as input an image of size 30x30 pixels, followed by a convolutional layer of 32 filters (in green). Each filter has 5x5 weights, with a stride of 1 pixel and

[1]The source code is available from the author upon request via email (eliveltoebermam@gmail.com)

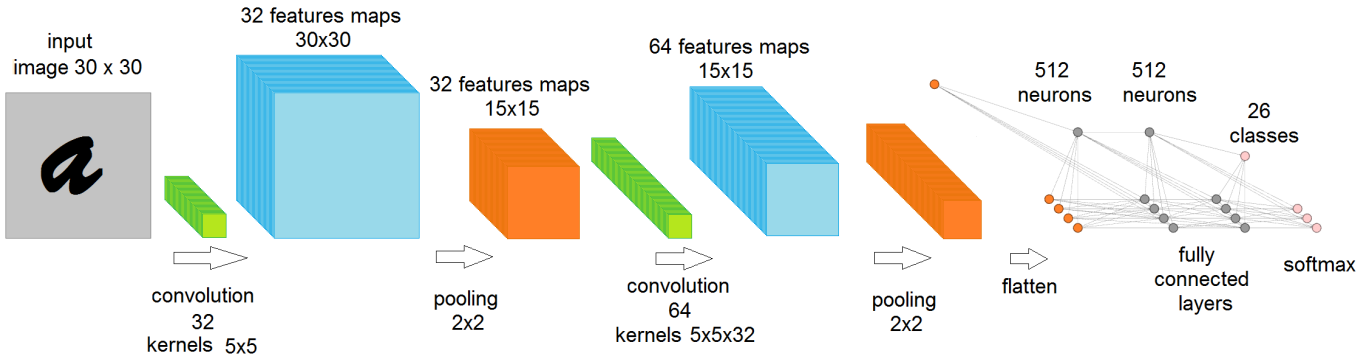

Fig. 10. Architecture of the convolutional neural network used in this case study

padding of 4 pixels to keep the original image size. Hence, we generate 32 feature maps of 30x30 pixels (in blue).

In each one of these feature maps we apply the max pooling operation of size 2x2 pixels, reducing both dimensions of each one of the maps by half. As a result, we have a new image of 15x15x32 pixels (in orange), i.e., an image of 32 channels.

In this image, a new convolution operation is performed with 64 filters with dimension 5x5x32, with the same values of stride and padding as the first one. As result, we have 64 feature maps of size 15x15 pixels, and each one of them goes through the max pooling operation. After that, we have an image of size 8x8x64 pixels, which is vectorized to a single dimension of 4096 pixels.

Each pixel is the input to a fully connected first layer with 512 neurons (in grey). We use dropout with 50% probability of signal propagation. The outputs of the first layer neuron is connected to another one of 512 neurons and once again we use the same droupout.

Finally, there is a layer with 26 neurons (in pink), which is followed by a softmax function, which calculates the probability of each class.

For the training, the images were scrambled and we used batches of 52 images with 2 images for each class in them. The activation function used was the Rectified Linear Unit (ReLU). The learning method was the stochastic gradient descent (SGD) with 0.9 momentum and learning rate of 0.01.

C. Simulation results

We performed 30 tests in a computer with a AMD Phenom X2 processor and 4 Gigabytes of RAM memory. The code was developed in Python with the use of the TensorFlow API. The network achieved accuracy of 91.08% of correct classifications on average, with standard deviation of 1.44%. The best result achieved was 93.1%. The network training took on average 1761.52 seconds, with a standard deviation of 118.09 seconds.

Table I shows the confusion matrix for the letters. The columns represent the correct letters and the lines, the network responses. A confusion matrix, or error matrix, is the most efficient way to represent the accuracy of each category, together with errors [31].

TABLE I
CONFUSION MATRIX OF THE DATABASE OF ALPHABETIC CHARACTERS

	a	b	c	d	e	f	g	h	i	J	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
a	59		2				2						1		1			1			2					1
b		66					1														1					1
c	1		65		1										2											
d				63	2			1				1					1				1					
e			1		64		1													1			1			1
f					1	61		2								1		2		1					1	
g	1				2		61										2				1	1				1
h	2							63						1			1	1							1	
i			1						53	3		9						1	1	1						
J		1							2	65												1				
k		2						1			61	1						1				1		2		
l	1				1				20	1		45						1								
m					1			1					65	2												
n		1						3						60				2				1		2		
o		1		2											64		1	1								
p				1												68										
q							2								2		64				1					
r	1										1						1	63	1	1					1	
s	1		2				1			1									64							
t						1												1		66		1				
u								2									1		1		63		2			
v															1						4	60	1		3	
w						1															1		67			
x	1				1												1							63	3	
y							1		1	1							1				2				63	
z	1				2							1														65

For most letters, the network found a good classification. The letters that were easier for the network were: p (98.5%), w (97.1%), b (95.65%) and t (95.65%). On the other hand, the letters that imposed the greater difficulty were: l (65.21%), i (76.81%), a (85.5%), n (86.95%) and v (86.95%). The main confusions committed by the network were exchanging l by i (28.98%), i by l (13.04%), v by u (5.79%), i by j (4.34%), n by h (4.34%) and v by y (4.34%).

Based on the fact that we presented both lowercase and uppercase letters, the possibility of having similar letters is higher, such as in the case of an "I" (uppercase i) with a "l" (lowercase L). Sometimes, a letter from a specific font is very hard to recognize and very easy to confound in isolation, and people usually distinguish them based on their context.

D. Comparison with another method

The results obtained from the CNN were compared with another neural network that has been widely used, the extreme learning machine (ELM) [32], [33].

The experiments performed for the ELM were done exactly like those for the CNN. The ELM used has 900 input units (one for each pixel in the image), 3000 neurons in the hidden layer and 26 neurons in the output layer. The activation function used was the sigmoidal function.

The ELM obtained an average classification accuracy of 77.03% with standard deviation of 0.75%. This network took in average 51 seconds to be trained, with a standard deviation of 5.88 seconds. Figures 11 and 12 show the box-plots for the classification accuracy for and the computational training time, respectively for both CNN and ELM.

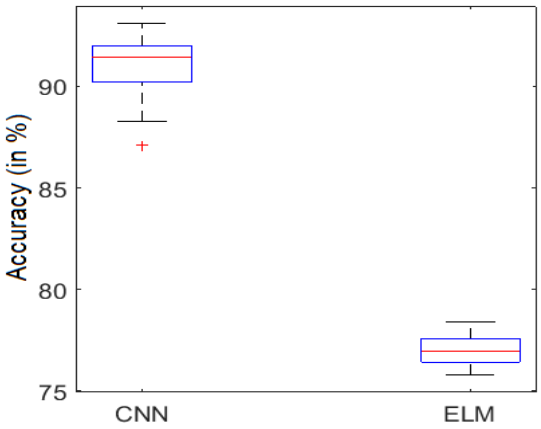

Fig. 11. Classification accuracy for test data in the alphabetic characters database

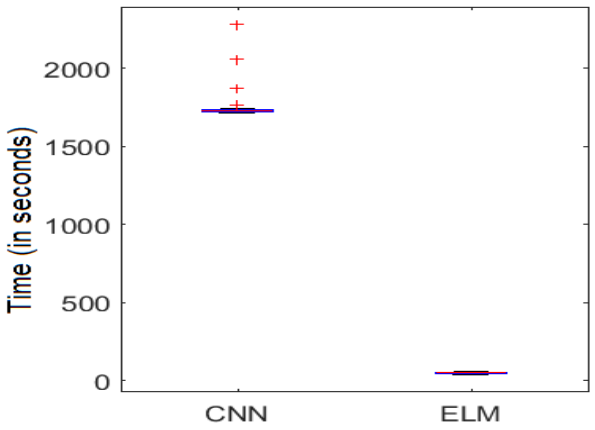

Fig. 12. Computational time to train the neural network

E. Evaluation of the performance of the convolutional neural network

The convolutional neural network, acting on the local receptive fields of the image, can preserve important information, such as the correlation between neighbouring pixels in the image. This is an advantage that it has over other neural networks, such as ELM, which need to change the image to one dimension, losing the original image structure. This way, as seen in the case of alphabetic characters classification, a CNN achieves a higher classification accuracy.

The CNN structure is also effective for multiple layers. The shared weights and pooling techniques reduce the number of weights in the network. Other neural networks of the type feedforward have difficulties with the training process by adding layers.

Nevertheless, the time to train a CNN is high. For the case study, it took 30 times longer than the ELM. Even with weight reduction techniques, the number of connections is still high. Training in an iterative way is a computational burden. Some CNN models require an adequate hardware to train (usually GPUs of the latest generation).

VI. Final Considerations

In this paper we presented a comprehensive introduction to convolutional neural networks in an intuitive and self-explanatory way. The network mechanism were explained in details, with examples that illustrate the processes. In addition, we presented a case study of the application of a CNN model for the classification of manuscript alphabetic characters and compared it with another neural network, the ELM.

The results for both techniques were described using their average and standard deviation both for classification accuracy and computational training time. The tests were performed 30 times. The classification accuracy of the CNN was a little inferior if compared to cases in which it was applied to the MNIST database, for instance. It should be taken into consideration the fact that the database used in this study presents some very difficult cases, such as the uppercase "I"and the lowercase "l", which are quite similar. Still, in terms of accuracy, the CNN was quite superior to the ELM.

We believe that modifying the network structure and other parameters, the results might be still better. However, the goal of this work was to show the CNN work principle in a simple and direct way, and for that we applied a standard model (LeNet). So, we expect that beginners in the area of computational intelligence might be able to use this material to better understand convolutional neural networks.

As future goals, we can extend the application of the CNN for words and license plates recognition. Another expansion is the application to color images and the recognition of animals, plants and objects.

Acknowledgements

E. Ebermam would like to thank CNPq for his M.Sc. scholarship. R.A. Krohling would like to thank CNPq and FAPES (Research Support Foundation for the State of Espirito Santo) for the financial support, given by grants No. 309161/2015-0 and No. 039/2016, respectively. The authors would like to thank the students Gabriel Giorisatto, André Georghton Cardoso Pacheco, Carlos Alexandre Siqueira da Silva and Igor de Oliveira Nunes for the creation and for making available the "Alphabet"database. The authors would also like to thank the reviewers and the editor in chief of this journal for their suggestions for improving the quality of the manuscript.

References

[1] Z. Bingul, H. M. Ertunc, & C. Oysu. Applying neural network to inverse kinematic problem for 6R robot manipulator with offset wrist. In *Adaptive and Natural Computing Algorithms*, pp. 112-115. Springer, Vienna. 2005.

[2] Y. LeCun, et al., Generalization and network design strategies, *Connectionism in perspective*, 1989, pp.143–155.

[3] Y. Lecun, Y. Bengio, D. Henderson, A. Weisbuch, H. Weissman, L. Jackel. On-line handwriting recognition with neural networks: Spatial representation versus temporal representation, Ecole Nationale Superieure des Telecommunications, 1993.

[4] K. O'Shea, R. Nash, An introduction to convolutional neural networks, arXiv preprint arXiv:1511.08458. 2015.

[5] J. Wu, Introduction to convolutional neural networks, National Key Lab for Novel Software Technology, Nanjing University, China. 2017.

[6] F. H. Araújo, A. C. Carneiro, R. R. Silva, Redes neurais convolucionais com tensorflow: Teoria e prática. III Escola Regional de Informática do Piauí. Livro Anais - Artigos e Minicursos, v. 1, n. 1, 2017, pp. 382-406.

[7] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, in *Proceedings of the IEEE*, vol. 86, n. 11, pp.2278–2324, nov. 1998.

[8] E. Mohebi & A. Bagirov. A convolutional recursive modified Self Organizing Map for handwritten digits recognition. *Neural Networks*, vol. 60, 2014, pp. 104-118.

[9] J. Bai, Z. Chen, B. Feng, B. Xu, Image character recognition using deep convolutional neural network learned from different languages, in: *IEEE International Conference on Image Processing (ICIP)*, 2014, pp. 2560–2564.

[10] K. A. Hamad, M. Kaya, A detailed analysis of optical character recognition technology, *International Journal of Applied Mathematics, Electronics and Computers*, vol. 4, Special Issue-1, 2016, pp. 244-249.

[11] R. Hecht-Nielsen, et al., Theory of the backpropagation neural network., *Neural Networks 1 (Supplement-1)*, 1988, pp.445–448.

[12] T. Fukuda, T. Shibata, M. Tokita, T. Mitsuoka. Neural network application for robotic motion control-adaptation and learning, In *IEEE International Joint Conference on Neural Networks*, vol. 2, 1990, pp. 447-451.

[13] J. Schmidhuber, Deep learning in neural networks: An overview, *Neural Networks* 61, 2015, pp. 85–117.

[14] W. S. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity, *The Bulletin of Mathematical Biophysics* 5 (4), 1943, pp. 115–133.

[15] S. Haykin, Redes Neurais: Princípios e Práticas, Ed. Bookman, 2001.

[16] A. Solazzo, E. D. Sozzo, I. De Rose, M. D. Silvestri, G. C. Durelli and M. D. Santambrogio. Hardware design automation of convolutional neural networks. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2016, pp. 224-229.

[17] D. H. Hubel, T. N. Wiesel, Receptive fields of single neurones in the cat's striate cortex, *The Journal of Physiology* 148 (3), 1959, pp. 574–591.

[18] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, vol. 36, 1980, pp. 193-202.

[19] D. E Rumelhart, G.E. Hinton & R. J. Williams, Learning internal representations by backpropagating errors. In *Nature*, vol. 323, 1986, pp. 533–536.

[20] Y. LeCun, Y. Bengio, Word-level training of a handwritten word recognizer based on convolutional neural networks, in: *Proceedings of the 12th IAPR International Conference on Pattern Recognition*, Vol. 2, IEEE, 1994, pp. 88–92.

[21] Y. Lecun, Y. Bengio, Convolutional networks for images, speech, and time-series, MIT Press, 1995.

[22] A. Krizhevsky, I. Sutskever, G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 2012, pp. 1097-1105.

[23] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, *Nature* 521 (7553), 2015, pp. 436-444.

[24] J. Gu, et al. Recent advances in convolutional neural networks. arXiv preprint arXiv:1512.07108, 2015.

[25] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting., *Journal of Machine Learning Research*, vol. 15, n. 1, 2014, pp. 1929–1958.

[26] M. Abadi, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software disponível em tensorflow.org.

[27] François Chollet. Comma.ai, 2016. URL http://keras.io/

[28] Y. Jia. Caffe: An open source convolutional architecture for fast feature embedding. http://caffe.berkeleyvision.org/.

[29] R. Collobert, S. Bengio, and J. Mariethoz. Torch: a modular machine learning software library. Technical Report IDIAPRR 02-46, IDIAP, 2002.

[30] Y. Tang. Deep learning using support vector machines. CoRR, abs/1306.0239, v. 2, 2013.

[31] R. G. Congalton, A review of assessing the accuracy of classifications of remotely sensed data. *Remote Sensing of Environment*, vol. 37, 1991, pp. 35-46.

[32] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, Extreme learning machine: a new learning scheme of feedforward neural networks. In *IEEE International Joint Conference on Neural Networks*, vol. 2, 2004, pp. 985–990.

[33] G. B. Huang, Q. Y. Zhu, and C. K. Siew. Extreme learning machine: theory and applications. *Neurocomputing*, vol. 70, 2006, pp. 489–501.