UNIVERSITY OF CAMPINAS
INSTITUTE OF MATHEMATICS, STATISTICS AND SCIENTIFIC COMPUTING
DEPARTMENT OF APPLIED MATHEMATICS

Gabriel Borin Macedo

Project Title : Evolutionary algorithms to train morphological neural networks[*]

Monograph presented to the Institute of Mathematics, Statistics and Scientific Computing of the University of Campinas as part of the crediting requirements for the discipline Supervised Project, under the guidance of Prof Dr João Batista Florindo.

Campinas
11/27/2019

[*]This work was funded by Serrapilheira Institute (grant number Serra-1812-26426).

Abstract

Numerous neural networks have been proposed in the literature for the task of image recognition. Among them, the *convolutional* neural networks trained by *backpropagation* are the most usually recommended for this task, as they frequently obtain excellent accuracy in a few iterations. However, during the last years, *morphological* neural networks have also been proposed as an alternative to the classical networks. Nevertheless, the non-differentiability of the convolution operator makes necessary the search for other training algorithms. In this context, evolutionary algorithms have arisen as a suitable strategy for training these types of networks. This project implements a *morphological* network to address the task of classifying manuscript numbers between 0 and 9 and compares the classification accuracy with that of a usual *convolutional* neural network. The accuracy obtained by the *morphological* network, using the *MNIST* benchmark database, reveals that *convolutional* neural networks still are the most appropriate solution for image classification. On the other hand, *morphological* neural networks are more flexible, allowing for example infinite possibilities for the definition of the convolution operator.

1 Introduction

The advent of high performance hardware, especially graphic cards (GPUs), allowed the application of *convolutional* neural networks (CNNs) in numerous scientific and industrial areas, making possible, for example, notorious advances in the task of localization, classification and segmentation of objects in images in the literature. These advancements also made possible to archieve important results in applications such as autonomous cars [1], games like chess and go [9], medical diagnostics [3], and many others. On the other hand, *morphological* neural networks have an interesting and differentiated approach to perform an image classification. That means that *morphological* networks trained with any evolutionary algorithm can do the same work as a *CNN* and, consequently, can substitute this kind of net.
To illustrate the idea that *morphological* neural networks can replace *CNN*s, two networks were implemented and tested in this project: one net was a *morphological* network trained by using the *Gravitational Search Algorithm*(GSA) (evolutionary) and the other one was a *convolutional* neural network trained by the classical *backpropagation* algorithm. These two networks were trained to classify hand written numbers between 0 and 9.
Based on these tests, this project investigated the performance of a model of *morphological* neural networks trained by *GSA* and compared the accuracy of this model with that of *convolutional* networks in the task of image classification.

2 Neural Networks

Neural networks are mathematical models used to predict and classify a given data (input), this input can be numbers, words, image, csv files with multiple tables, among other examples. The operations of this network are very simple and composed by neurons that receive and process an input data $x_1, x_2, ..., x_n$. In the most basic version, usually called perceptron, the essential processing consists of a linear combination of the inputs, followed by the application of a non-linear *activation function* ξ, which at the end provides a scalar number Z as the final result (output). To be more didactic on this work, it will be adopted ξ as *f(v)*. The image bellow represents this idea.

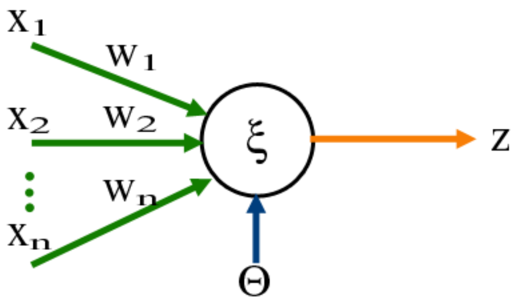

Figure 1: Generic representation of a basic neural network (perceptron)

and

$$v = f(\sum_{i=1}^{n} x_i w_i + \theta), \tag{1}$$

where w_i is the weight associated with each neural connection and θ is an arbitrary constant. The structure above is composed by one neuron (image 1) and is called a *perceptron* in the literature. θ is a parameter called *bias*.
Such bias is a measure of how easy is for each perceptron to "*activate*". For example, a binary perceptron that outputs only 1 or 0 (step function) and has large bias has higher probability of producing an output 1.
The image bellow represents a step function :

$$f(x) = \begin{cases} 0 & \text{if Eq(1)} \leq 1 \\ 1 & \text{otherwise} \end{cases}$$

2.0.1 Softmax Neurons

By stacking groups of perceptrons into layers, it is possible to create a network of perceptrons. This network can be used to solve a determined problem by automatically "learning" weights and biases to produce a correctly output or very close to the answer predicted by the data used for training. A good principle for the learning process is that making a small change in some weight or bias should cause only a small corresponding change in the output of the network.
It is possible to use this principle to modify the weights and biases of neural networks. This method of changing the weights and biases repeatedly in a limited number of interactions to produce better and better output is what is called *learning*.

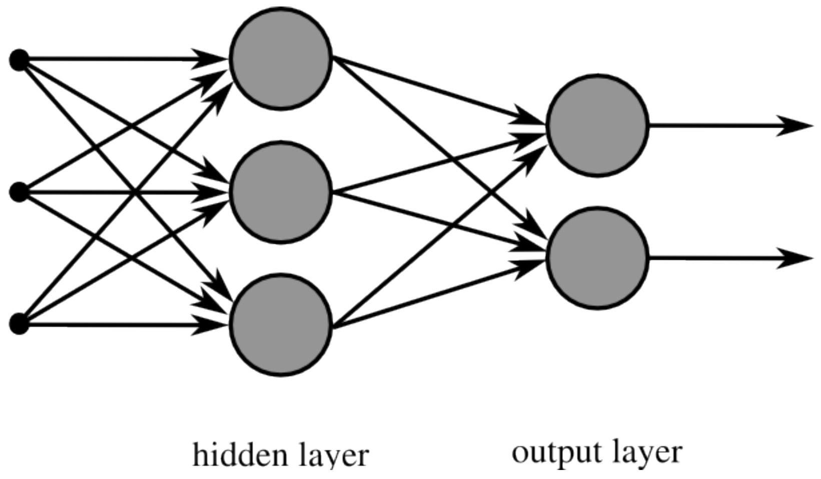

Figure 2: Generic representation of a multi-layer neural network.

However, networks that have only binary perceptrons are not sufficiently robust and may have their output easily modified. For example, an output 0 might be changed to 1. This flip can cause the behaviour of the rest of a network to change dramatically in a very complicated way, preventing the network from performing correct classification. Besides, this process makes difficult to see how the weights and biases are gradually being modified along the training and if the network is really getting closer to the desired behaviour.
Based on this context, softmax neurons can be used as they are more consistent with changes of weights and biases, resulting in more concise and precise classification. Softmax neurons have similar behavior to binary perceptrons, i.e., the neurons also receive an n-dimensional input $x_1, x_2, ..., x_n$. Nevertheless, the output can be any real value between 0 and 1, instead of exactly 0 or 1.
The softmax activation function is also of great importance in these neural models, as their outputs behave like probabilities for the potential outcomes. The equation below represents this idea.

$$y = [y_1, y_2, ..., y_i, ..., y_n] = S(y_i) = \frac{\exp y_i}{\sum_{j=1}^{n} \exp y_j} = V_p = [p_1, p_2, ..., p_i, ..., p_n]$$

$$\sum_{k=1}^{n} p_k = 1$$

where y_i refers to each element in the output vector y and V_p is the vector of probabilities.
The bias expression is :

$$b = \frac{1}{1 + e^{(-v)}} \tag{2}$$

where v comes from from Eq. (1).
The smoothness of $S(y_i)$ implies that a small change in the weights δw_i and biases δb will produce small change δ_{out} in the neuron output. The equation below represents this idea :

$$\delta_{out} \approx \sum_i \frac{\partial out}{\partial w_i} \delta w_i + \frac{\partial out}{\partial b} \delta b \tag{3}$$

where the sum includes all w_i weights. $\frac{\partial out}{\partial w_i}$ and $\frac{\partial out}{\partial b}$ are partial derivatives of the output with respect to δw_i and δb, respectively. Lastly, δout is a linear function of the changes of weights (δw_i) and biases (δb).
These neurons play important role in the mechanism of *fully connected* networks [8].

3 Training Neural Networks using the *Gravitational Search Algorithm*(GSA)

The most popular algorithm to update the weights and biases over the training step is the *backpropagation* [5]. However, a fundamental requirement of backpropagation is that all the functions involved in the network processing should be differentiable. This is not possible in some situations and investigating algorithms that could be applied in mode general situations, including non-differentiable functions, is an important matter.
Based on this, here study the replacement of the usual algorithm of *backpropagation* by the *Gravitational Search Algorithm* (GSA). Most optimization algorithms in *machine learning* are created to find the best "answer" to a problem by minimizing some type of "cost function". In more practical terms, such "best answer" correspond to the weights that minimizes the classification error and maximizes the network accuracy or. It is important to emphasize that situations of *local minimum* may affect negatively the precision and, consequently, this network can make a prediction or a classification with a poor accuracy. GSA is part of family of optimization methods, called evolutionary algorithms, that use heuristics to achieve the minimization. Other examples of algorithms in this family are *Particle Swarm Optimization* (PSO) [7], *Ant Colony Optimization* (ACO) [2], and many others.
Based on this problem that have a set of solutions, the *Gravitational Search Algorithm* proposes a method to find the global minimum by making each solution having n components and interacting with each others using a function. This approach and the work [4] model this sets of solutions as masses with n variables that interact with each others using the *gravitational force*. The set of masses is defined by:

$$X_i = (x_i^1, x_i^2, ..., x_i^d, ..., x_i^n),$$

where $i \in [1, N]$, N is the number of solutions, n is the index of the variables (as defined before) and x_i^d is the position of the solution i in the dimension d.
The gravitational force of interaction is defined by:

$$F_{ij}^d(t) = G(t)\frac{M_{pi}(t) * M_{aj}(t)}{R_{ij}(t) + \epsilon} x(x_j(t) - x_i(t)), \tag{4}$$

where $M_{aj}(t)$ is the "mass" of the solution j, $M_{pi}(t)$ is the "mass" of solution i, $G(t)$ is the function that calculates the gravitational constant and $R_{ij}(t)$ is the function that calculates the Euclidean distance between the solutions i and j.
The expression of $G(t)$ and $R(t)$ are defined as follows:

$$G(t) = G_0 * e^{-\alpha * \theta(t)} \tag{5}$$

and $\theta(t)$ is : $\theta(t) = \frac{t}{T}$

$$R_{ij}(t) = \langle X_i(t), X_j(t) \rangle, \tag{6}$$

where $\langle \cdot, \cdot \rangle$ is the usual scalar product that maps two R^N vectors to R by doing the operation : $\langle X_i(t), X_j(t) \rangle = \sum_{k=0}^{N} \sum_{l=0}^{N} X_{ik}(t) * X_{jl}(t)$

The α term in Equation (6) is the coefficient, G_0 is the gravitational constant (the same used in the gravitational force on physics), t is the current iteration and T is the maximum number of iterations. In this method, the gravitational forces applied between one solution and the others are calculated as in Eq.(7)

$$F_i^d(t) = \sum_{j=1, j\neq i}^{N} rand_j * F_{ij}^d(t), \tag{7}$$

where $rand_j$ is a real number in the interval [0, 1]. After this, this algorithm will calculate, in this order, the acceleration and velocity applying the formulas:

$$a_i^d(t) = \frac{F_i^d(t)}{M_{ii}(t)}, \tag{8}$$

where d is the number of variables in the problem , t is the current iteration and $M_{ii}(t)$ is the inertial mass of the agent i.

$$v_i^d(t+1) = rand_i * v_i^d(t) + a_i^d(t) \tag{9}$$

Finally, the position of every solution is updated by using the expression in Eq (10):

$$x_i^d(t+1) = x_i^d(t) + v_i^d(t) \tag{10}$$

It is important to notice that each one of the solution is the fitness value computed by the fitness function [6]. Thereby, every solution tend to become proportional to the value of the fitness function. It is possible to easily attenuate this issue, caused by the direct relation between mass and the fitness function, by adopting a normalization of these terms. The normalization method is defined as follows:

$$M_i(t) = \frac{m_i(t)}{\sum_{j=1}^{N} m_j(t)}, \tag{11}$$

where $m_i(t)$ is the expression :

$$m_i(t) = \frac{fit_i(t) - worst(t}{best(t) - worst(t)},$$

where $fit_i(t)$ is the fitness value of the solution i in the interaction t, *best(t)* is the best solution in the interaction t and *worst(t)* is the worst solution in the interaction t. In the first iteration of GSA, all the variables are initialized with random real values and the GSA stops when it meets the stopping criterion.
Furthermore, this method has a slow convergence speed to find the best result and stays susceptible to be trapping in a local minima. These issues are due to the effect of the fitness function when calculating any mass (solution). Besides, the search agents get more heavier on each interaction. Thus, causing a decrease of the speed search. This complication prevents the search agents from discovering other minimum locals, which could be candidates to be global minimum. With that, a *chaotic* method is proposed to tackle this scenario.

3.0.1 *Chaos Theory* and *Chaotic maps*

Chaos Theory refers to the study of *chaotic dynamical systems*, which are nonlinear dynamical systems highly sensitive to their initial conditions. In other words, when the system has a small change in the initial conditions, it may result in high variation in the output of the system. The apparently random behavior of chaotic systems is an interesting feature. It is important to emphasize that a system does not necessarily need to be random to present chaotic behavior. This means that *deterministic* systems can also present chaotic behaviors.
This work will use the same *chaotic maps* utilized on [4].

3.1 Normalization and updating the weights

In this stage of the studied algorithm, we define the expression $I(t) = MAX - \frac{t}{T} * (MAX - MIN)$ to calculate the normalization of $C_i(t)$ that goes from [a, b] to [0, $I(t)$]. The formula of each new $C_i(t)$ after normalization is:

$$C_i^{norm}(t) = \frac{(C_i(t) - a) * (I(t) - 0)}{(b - a)}, \tag{12}$$

where i is the index of the chaotic map, t is the current iteration, T is the maximum number of iterations, *[MAX, MIN]* is the adaptive interval and [a, b] is the range of the chaotic map.
At the end, the final value of the gravitational constant is updated as follows :

$$G(t) = G_0 * e^{-\alpha * \theta(t)} + C_i^{norm}(t), \tag{13}$$

$$\text{with } \theta(t) = \frac{t}{T}$$

where α is the descending coefficient, G_0 is the initial gravitational constant, t is the current iteration, T is the maximum number of iterations and i is the index of the chosen chaotic map.
This pseudo code of GSA using chaotic maps is a summary of this method :

Algorithm 1 GSA with Chaotic Maps

```
procedure Initial Iteration
    Generate initial population.
    Compute the fitness for all search agents.
loop:
    if (the end criterion is satisfied) then
        return the best search agent.
    Update G(t) using Eq.(3.1).
    Update M, forces, and accelerations.
    Update the velocities and positions.
    continue the loop.
```

4 Morphological Neural Networks (MNN)

In the general theory of neural networks, *Morphological Neural Networks* (MNN) have a peculiar behavior compared with usual *convolutional* nets. Instead of using the Eq. (2) to calculate the output on each node, these nets use the following expression for the node output:

$$v_j = f_j(\vee_{i=0}^{N} x_i + w_i). \tag{14}$$

In this model, the algebra space where the operations are applied is the extended real numbers, defined by $\mathbb{R}_{\pm} = \mathbb{R} \cup (-\infty, \infty)$. The operator $\vee$ is replaces the usual sum. The value of $-\infty$ and ∞ have the same function as 0 and 1, respectively, in the classical algebra of sums and products and f_j is the activation function in the j-node.

4.1 Morphological Operators

To better understand the behavior of *morphonets*, this topic will describe some operands and operations defined in *image algebra* that are useful for this type of neural network.

4.1.1 Mathematical definition of *morphological* operators

In more mathematical terms, an image is described as a function mapping a subset $\mathbf{X}$ of $\mathbb{R}^{\mathrm{N}}$ onto a set of discrete values, denoted by $\mathbb{F}$. Thus a generic element $a \in \mathbb{F}^{\mathbf{X}}$ and we use the notation [(x, a(x)) : x $\in \mathbf{X}$, a(x) $\in \mathbb{F}$]. It is also possible to describe any image using a finite subset $\mathbf{X}$ with q elements that will have the following structure : [(i, a(i)) : i=$1, 2, \ldots, q$, where a(i) $\in \mathbb{F}$]. A *template* is an element of $(\mathbb{F}^{\mathbf{X}})^{\mathbf{Y}}$, where $\mathbf{X} \in \mathbb{R}^{\mathrm{N}}$ and $\mathbf{Y} \in \mathbb{R}^{\mathrm{M}}$. Another alternative formulation to describe the *template* consists in defining $\mathbf{t}$ as a function t: $\mathbf{T} \to \mathbb{F}^{\mathbf{X}}$ and having the form [(y, [x, $t_y(x)$]) : y $\in \mathbf{Y}$, [x, $t_y(x)$] $\in \mathbb{F}^{\mathbf{X}}$].
For notation convenience, we use the expression t_y instead of t(y). It is important to notice that $\mathbf{t}$ is a function that associates every point $\mathbf{y}$ of the domain to some image element $t_y \in IF^{\mathbf{X}}$.

4.1.2 Definition of operations between images and templates

It is also possible to combine the image and template in an operation. In this case, the computation only needs to take place over the support of the template. Formally, the *infinite*-support $S_{-\infty}(t_y)$) for a template $\mathbf{t} \in (\mathbb{R}^{\mathbf{N}}_{\pm\infty})^{\mathbf{M}}$ is the set of pixels in $\mathbf{X}$ where $t_y(x) \neq -\infty$ and $S_{-\infty}(t_y) = \{\mathrm{x} \in \mathbf{X} : t_y(x) \neq -\infty\}$.
Two typical supports used are the *von Neumann* or *4-neighborhood* (that are the closest pixels from up, down, left and right) of the pixel $\mathbf{y}$ and the *Moore* or *8-neighborhood* (the eight closest pixels to the pixel $\mathbf{y}$).
Thus three basic operations are defined between two images $\mathbf{a}$, $\mathbf{b} \in \mathbb{R}^{\mathbf{X}}_{\pm\infty}$ that produce a new image $\mathbf{c} \in \mathbb{R}^{\mathbf{X}}_{\pm\infty}$.

$$a + b = c = \{(\mathbf{x}, \mathbf{c}(\mathbf{x})) \ : \ \mathbf{c}(\mathbf{x}) = \mathbf{a}(\mathbf{x}) + \mathbf{b}(\mathbf{x}) \ , \ \mathbf{x} \in \mathbf{X}\}$$

$$a * b = c = \{(\mathbf{x}, \mathbf{c}(\mathbf{x})) \ : \ \mathbf{c}(\mathbf{x}) = \mathbf{a}(\mathbf{x}) * \mathbf{b}(\mathbf{x}) \ , \ \mathbf{x} \in \mathbf{X}\}$$

$$a \vee b = c = \{(\mathbf{x}, \mathbf{c}(\mathbf{x})) \ : \ \mathbf{c}(\mathbf{x}) = \mathbf{a}(\mathbf{x}) \vee \mathbf{b}(\mathbf{x}) \ , \ \mathbf{x} \in \mathbf{X}\}$$

It is also possible to define operations between templates **s** and **t** $\in (\mathbb{R}^{\mathbf{X}}_{\pm\infty})^{\mathbf{Y}}$.

$$s + t = r, r_y(x) = s_y(x) + t_y$$

$$s * t = r, r_y(x) = s_y(x) * t_y$$

$$s \vee t = r, r_y(x) = s_y(x) \vee t_y$$

4.1.3 The *step function* and weight definition in *morphological* networks

Inspired by the behavior of the *step function*, we can define one function that produces 0 or $-\infty$ instead of 0 or 1 as output values :

$$f^{-\infty}_{>0}(a) = b, b(x) = \begin{cases} 0 & \text{if a(x)} > 0 \\ -\infty & \text{otherwise} \end{cases}$$

In a similar manner, this function can be applied to a template. The comparison $>$ can also be replaced with any other, such as $=$, $\leq$, and others.
Finally, the weights are described by relating the number of nodes and the number of exemplar patterns. Let N be the number of nodes in the net, P be the number of exemplar patterns and $\mathbf{X} = \{1, 2, \ldots, N\}$. The weights are determined by setting :

$$w_{ji} = t_j(i) = \begin{cases} \sum_{k=1}^{P} x^k_j * x^k_i & \text{if i} \neq \text{j} \\ 0 & \text{otherwise,} \end{cases}$$

where x^k_j is the j-th element of the exemplar pattern from class k.

4.1.4 The definition of dilatation operator

There exist two types of morphological operators that are frequently adopted on image processing, especially to analyze object shapes on images. They are essential to realize the math operations on each node. The first one is named *dilatation*. This operator is defined as follows :

$$a \vee t = b = [(y, b(y)) : b(y) = \vee^{n}_{i=0} a(x_i) + t_y(x_i), \text{with } y \in Y] \tag{15}$$

This operation uses $\mathbb{F} = \mathbb{R}_{\pm\infty}$ and is important to describe the operation on each node in a *morphological* neural network. The input image **a** is composed by a finite set of discrete values (consequently they do not assume values $-\infty$ or ∞). To maintain the consistency of this model, it is necessary to define the addition operation between all the elements of $\mathbb{R}_{\pm\infty}$. This can be trivially solved by setting the operations using the usual operator of addition $+$:

$$\text{a} + (-\infty) = (-\infty) + \text{a} = -\infty$$
$$\text{a} + \infty = \infty + \text{a} = -\infty$$
$$(-\infty) + \infty = \infty + (-\infty) = (-\infty),$$

where $\mathbf{a} \in \mathbb{R}_{-\infty} = \mathbb{R} \cup [-\infty]$. However, here we assume that all the input images $\mathbf{a} \in \mathbb{R}_{\pm\infty}^{\mathbf{X}}$ are composed by a set of finite and discrete values, which means that image $\mathbf{a} \in \mathbb{R}^{\mathbf{X}}$. If $\mathbf{a}$ is a binary image, $\mathbf{a}$ has only the numbers 0 and 1. If $\mathbf{t}$ is a *binary template*, $\mathbf{t}$ assumes only the values 0 and $(-\infty)$.
As mentioned earlier, the main characteristic that differentiates *morphological neural networks* from *classical neural networks* is the inclusion of the values ∞ and $-\infty$. Furthermore, *morphological* nets use values $\in \mathbb{R}_{\pm\infty}$. In image processing, all the template values outside that support are ignored. Consequently, to make valid this idea, those values must be set to $-\infty$. With this elements, it is possible to construct a simple morphology neural net that uses the *step function* as activation function, the image $\mathbf{a}$ as input, the template $\mathbf{t}$ and the Eq. (4) in every node.
In general, the basic calculation in a *morphological* neural network can be expressed as :

$$\text{f(a} \vee \text{b)} = (f_1(a \vee b),\ f_2(a \vee b),\ \ldots,\ f_N(a \vee b))$$

For example, for a neural net that has 9 nodes in total, the seventh node value at the upper layer of the network can be found by calculating the follow expression :

$$b_7 = \vee_{i=1}^{n=9} a_i + w_{7\text{i}} = \vee_{i=1}^{n=9}\{a_1 + w_{71}, \ldots, a_1 + w_{79}\}$$

$$= \vee_{i=1}^{n=9}\{9 + (-\infty), \ldots, 4 + (-\infty), 3 + 1, 2 + 2, 1 + (-\infty)\}$$

$$= \vee_{i=1}^{n=9}\{(-\infty), \ldots, (-\infty), 4, 4, (-\infty)\} = 4$$

5 Experimental Result

To evaluate the performance of *morphological* neural networks trained with *GSA*, the dataset of *MNIST* (manuscript digits between 0 and 9) was chosen to assess the classification performance. In addition, a classical *CNN* was trained with the same dataset. These two networks were implement using *Python*, but only the API of *Keras* (package of *Python* to create general neural networks) was used to create the second network. The training stage was accomplished in 35 epochs, where each epoch had a set of 1000 images. The weights were updated by using 500 images in the training set. Compared with *CNN*, *morphological* networks had poor accuracy in the classification of each digit. In the first training epoch, the error was around at 75 % and was increasing on each update iteration. At the end of the training, the error was around 90%, and consequently, the model accuracy was around 10%. The average computational time for each epoch was around 40 minutes.
On the other hand, the *CNN* shows a better accuracy in a few epochs and, experimentally, proved to be faster than the *morphological* neural networks in each training epoch. Below, it is possible to see, respectively, the error plot in each epoch of the *morphological* network and the accuracy curve in each epoch of the *CNN*.

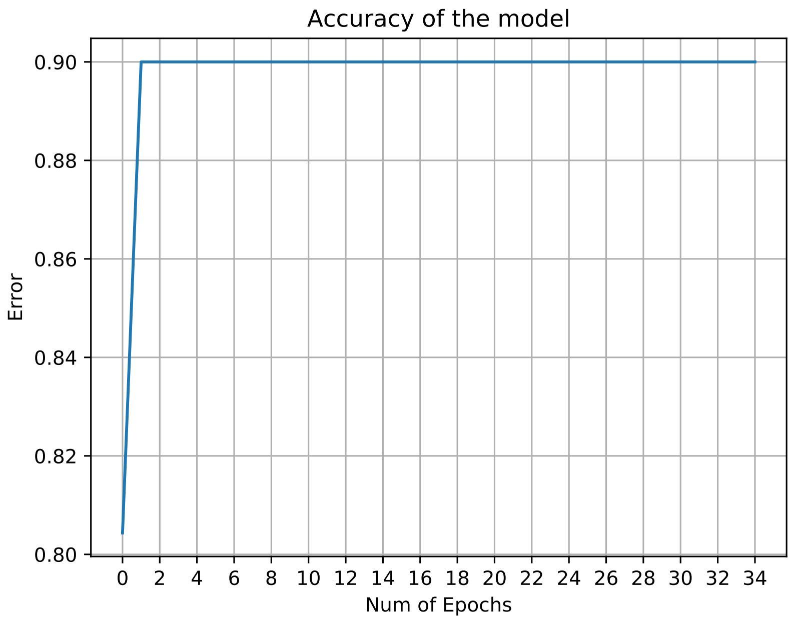

Figure 3: Plot of the error of the *morphological* network in each epoch.

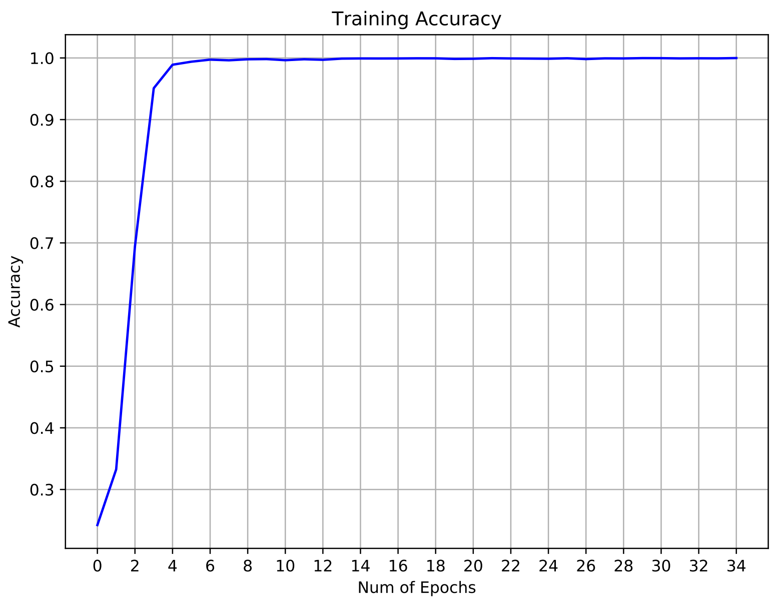

Figure 4: Plot of the accuracy of the *CNN* in each epoch.

6 Conclusions

The results of the experimental tests demonstrate that *morphological* neural networks are not as efficient as *CNN*s. The final accuracy of the *morphological* neural network was around 10% while the final accuracy of the *CNN* was around 90%. Besides that, each epoch of training was computationally more expensive, consequently, took more time to train each epoch. Based on this, we suggest that *CNN* can be a preferred over *morphological* neural network to perform image classification. We intend to investigate other alternatives that make the use of MNNs viable, as they still have the advantage of being considerably more flexible and generalist than CNNs.

References

[1] B. Chen, C. Gong, and J. Yang. Importance-aware semantic segmentation for autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 20(1):137–148, 2019.

[2] Jing Tian, Weiyu Yu, and Shengli Xie. An ant colony optimization algorithm for image edge detection. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 751–756, June 2008.

[3] Yiming Liu, Pengcheng Zhang, Qingche Song, Andi Li, Peng Zhang, and Zhiguo Gui. Automatic segmentation of cervical nuclei based on deep learning and a conditional random field. *IEEE Access*, PP:1–1, 2018.

[4] Seyedali Mirjalili and Amir H. Gandomi. Chaotic gravitational constants for the gravitational search algorithm. *Appl. Soft Comput.*, 53(C):407–419, April 2017.

[5] Varun Ranganathan and S. Natarajan. A new backpropagation algorithm without gradient descent. *CoRR*, abs/1802.00027, 2018.

[6] Esmat Rashedi, Hossein Nezamabadi-pour, and Saeid Saryazdi. Gsa: A gravitational search algorithm. *Information Sciences*, 179(13):2232 – 2248, 2009. Special Section on High Order Fuzzy Sets.

[7] Saptarshi Sengupta, Sanchita Basak, and Richard Alan Peters II. Particle swarm optimization: A survey of historical and recent developments with hybridization perspectives. *CoRR*, abs/1804.05319, 2018.

[8] Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(4):640–651, April 2017.

[9] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.