ISSN 1870-4069

Convolutional Model with Classification through Izhikevich Neuron

Antonio Luna-Álvarez, Dante Mújica-Vargas, Manuel Mejía-Lavalle

Tecnológico Nacional de México/CENIDET, Cuernavaca, Morelos, Mexico
{jesus.luna18ce,dantemv,mlavalle}@cenidet.edu.mx

Abstract. This paper presents the fusion of two paradigms of neural networks: the Convolutional Neural Networks from deep learning and the third generation Izhikevich neuron. This fusion has the purpose to replace the multilayer perceptron layers, that usually represent a computational cost and a large training time, for a paradigm created to classify with a single neuron. The experimentation is carried out in the classification domain, predicting the directions of rotation in a simulator of self-driven vehicles. The experiments show similar results to the multilayer perceptron model in the evaluation metrics but the training time is reduced.

Keywords: Izhikevich neuron, convolutional neural network, deep learning, steering classification.

1 Introduction

Deep Neural Networks is one of the most used techniques nowadays for the object detection and classification in digital images, because they do not require specific parameters for each type of image, they adapt to the image in a automatic. However, one of the great disadvantages of this paradigm is the slow training process, which even with hardware of great processing capacity, requires hours or days of processing. Another limitation is the need for large databases to train and be able to perform the task successfully.

To reduce the training time of a Deep Neural Network, several solutions have been proposed. One way to reduce time is to reduce the training data, a related paper is [4] in which it is proposed to make a selection of representative data with heuristic values granted by human experts, this method decrease the training time without affecting the performance of the neural network. There are also more intelligent proposals in which techniques accelerate the learning of the network. For example in [18] is proposed to use an optimization technique based on Particle Swarm for the backpropagation, decreasing training epochs. Another way to accelerate learning is by intelligently tuning of the training parameters using sophisticated techniques like the shown in [17], which presents an optimization of the learning rate, and likewise to the previous proposal, decreasing the training epochs.

A less sophisticated but effective alternative to reduce training time is to increase the hardware used for this task, for example in [7] a framework is created to distribute the processing in a cluster of servers. And on the other hand, the use of deep neural networks in embedded systems has been very difficult due to the need for powerful hardware, which has forced to improve the devices in which these systems are used, as in the work of [10].

Although the proposals of the state-of-the-art present acceptable results there are some complications. In the first place, using only representative data might cause the network not to be able to classify correctly those objects that are not very recurrent in the training data or that resemble the most common ones. On the other hand, although intelligent methods for learning acceleration are the ideal technique to reduce the training epochs, it implies a second processing and as is well known, the convergence of the Particle Swarm algorithm requires a long time due the complexity of $O(n \cdot p + Cof \cdot p)$ where n is the dimension of the problem p is the population size and Cof is the cost of the objective function. For this reason, although they reduce epochs, training time can be increased. Finally, enhancing hardware is not the best option when you do not have the necessary resources.

A solution proposal more to the problem is the implementation of a updated neuronal paradigm. In this document is proposed the fusion of a Convolutional Neural Network for the extraction of information from digital images and the Izhikevich Neuron [8] used as a control model. The combination of the deep learning paradigm and the third-generation neuron allows a reduction of the training time without using algorithms or additional methods to the training of this neuronal model. The fusion of these paradigms allows to reduce the number of neurons in the classification layers (multilayer perceptron) thus causing a reduction of the processing without affecting the performance in the classification.

The following sections of the document sections are organized as follows. The Section 2 describes the theoretical concepts necessary to understand the functioning of convolutional layers in a deep neural network. The description of the third generation Izhikevich neuron is shown in the Section 3, as well as the fusion of this with the convolutional layers. The data used for the experimentation, the methods of evaluation and the configurations of the experiments are presented in the Section 4. Finally the results obtained and the comparison is made in the Section 5 and the conclusions in the Section 6.

2 Background

2.1 Convolutional Network

This type of neural network has great advantages of automation in the task of extracting image information. In this paradigm, no known image processing filter is applied, but through the training, it learns the image processing filters. In general, the first convolutional layer learns to detect edges, while the second can learn to detect more complex shapes that can be formed by combining

different edges, such as circles and rectangles. The third layer and beyond learn much more complicated features based on the characteristics generated in the previous layer [14].

The Convolutional Neural Networks (CNN) are based on the convolution of signals in two dimensions and on the detection of filter-based characteristics (called kernel) that they learn through training. To extract the information, the kernel performs the 2D convolution operation, expressed in the Equation 1, to the input image. Where N and M are the dimensions of the image, n_1 and n_2 represent the column and row indices of the pixel being processed, k_1 and k_2 the kernel indexes [12]:

$$y(n_1, n_2) = \sum_{k_2=0}^{N-1} \sum_{k_1=0}^{M-1} x(k_1, k_2) h(n_1 - k_1, n_2 - k_2) \\ 0 \leq n_1 \leq N-1, 0 \leq n_2 \leq M-1. \tag{1}$$

To train the convolutional layers, the backpropagation of the error technique is used through the convolutional layers. This is very similar to the backpropagation for a multilayer perceptron network, the difference is that the weight connections are scattered, since the different input areas share the same weights to create a map of output characteristics. In general, the feature map is obtained through the function expressed in (2). Where w represent the kernel weights and a a first feature map obtained. On the other hand, the kernel weights are adjusted by calculating the descendant gradient which is expressed in (3). Where L represents the error function:

$$S_{ij} = \sum_{n=1}^{2} \sum_{m=1}^{2} w_{(3-m)(3-n)} \cdot a_{(i-1+m)(j-1+m)}, \tag{2}$$

$$\frac{\partial L}{\partial w_{ij}} = \sum_{j=1}^{2} \sum_{j=1}^{2} \frac{\partial L}{S_{ij}} \frac{\partial S_{ij}}{\partial w_{ij}}. \tag{3}$$

In the generality of the Convolutional Neural Networks, there are layers of dimensionality reduction; Max Pooling layers are given by a discriminant function where the maximum value is selected in each region of the image by moving a kernel through it, thus generating a new feature map of smaller dimensions to the original. The function of this filter is given in (4), where n is the kernel dimension, N is the dimension of the image and $u(\cdot)$ is the window function:

$$a_j = \max_{N \times N} (a_i^{n \times n} u(n, n)). \tag{4}$$

On the other hand, there exists layers of Average Pooling, which follows the same principle as the previous one, with the difference that in this, the kernel

take an average of the region according to (5) [16]:

$$a_j = \frac{\sum_{N \times N} a_i^{n \times n}}{N^2}. \tag{5}$$

2.2 Izhikevich Neuron

This model emulates the pulsation behavior of a certain type of neurons known as cortical. The model combines the biological plausibility of Hodgkin-Huxley-type dynamics and the computational efficiency of integration and firing neurons (spiking networks) [9]. This neuron has the ability to perform non-linear classification unlike its predecessor, the simple perceptron. The model in question has been evaluated in multiclass [15] and non-linear applications [13].

Algorithmically this neuron depends on a series of parameters, which are distributed in three equations [6]; (6) represents the energy gain of the neuron, in (7) recovers the energy through time and finally (8) emulates the pulse of the neuron to reach a certain threshold and the restart of the neuron gains [5]:

$$C\frac{dv}{dt} = k(v - v_r)(v - v_t) - u + I, \tag{6}$$

$$\frac{du}{dt} = a(b(v - v_r) - u), \tag{7}$$

$$v \geq v_{peak} \Rightarrow \begin{cases} v \leftarrow c, \\ u \leftarrow u + d. \end{cases} \tag{8}$$

The weight adjustment is one of the contributions of [5] to Izhikevich's original model. This adjustment is given by calculating the difference in pulses obtained and expected, multiplied by the inner product of the synaptic weights w and the inputs x, scaled by the learning rate α. However, this method works correctly for the binary classification, but for a classification of multiple classes it is necessary to make an extension of this function using the medium, as shown in (9):

$$\Delta w = \alpha \cdot \sum_{i=0}^{n} \frac{1}{n}(y - y')^2 \cdot (w \times x). \tag{9}$$

In general, an Izhikevich neuron is trained through N epochs, for each the calculation of synaptic currents I is made based on the inner product of the weights and the inputs plus the bias, this synaptic current serves input for the calculation of the membrane potential of the v neuron that accumulates through 1000 iterations, achieving y pulsations. The algorithm of a single epoch is defined as shown in Algorithm 1.

Algorithm 1 Izhikevich neuron algorithm.

Input: $x \in \mathbb{R}^n, y' \in \mathbb{Z}$
Output: $y \in \mathbb{Z}$
  $I \leftarrow [(w \times x) + 1]^2 + \theta$
  $u \leftarrow 0$
  $v \leftarrow v_r$
  for $i \leftarrow 0$ **to** $i = 1000$ **do**
    $v \leftarrow \frac{v+[k(v-v_r)-(v-v_t)-u+I]}{C}$
    $u \leftarrow d[b(v - v_r) - u]$
    if $v \geq v_{peak}$ **then**
      $v \leftarrow c$
      $u \leftarrow u + d$
      $y \leftarrow y + 1$
    end if
  end for
  $E(w) \leftarrow \frac{1}{n} \sum_{i=0}^{n} (y_i - y'_i)^2$
  $\Delta w \leftarrow \alpha \cdot E(w) \cdot (w \times x)$
  return y

Here $x \in \mathbb{R}^n$ represents the input vector, $y \in \mathbb{Z}$ the number of pulses emitted by the neuron and $y' \in \mathbb{Z}$ the expected pulses. $w \in \mathbb{R}^n$ represents the weights that match the inputs. The variable $v \in \mathbb{R}$ represents the membrane potential of the neuron and $u \in \mathbb{R}$ represents a variable of membrane recovery, which explains the activation of ionic currents $k \in \mathbb{R}$ and the inactivation of ionic currents, and provides negative feedback to v.

3 Convolutional Network and the Izhikevich Neuron Fusion

As mentioned in the Section 1, the use of another type of neuron than the perceptron in the classification layer reduce the training time. To extract information from digital images, it is still necessary to use a Convolutional Neural Network, however it is possible to replace multilayer perceptron layers, in this case by a spiking Izhikevich neuron.

In general, the fusion occurs when the numerical information obtained in the process carried out by the CNN is extracted. This numerical information is introduced as a synaptic vector to the Izhikevich neuron, similar to how it is done with the simple perceptron. The neuron performs the process mentioned in Algorithm 1 during training N epochs to adjust the weights w. Unlike the perceptron, the Izhikevich neuron does not return predictions given by the function $\sigma(\cdot)$ where $y' \in \mathbb{R} \rightarrow [0, 1]$ or by the function $tanh(\cdot)$ where $y' \in \mathbb{R} \rightarrow [-1, 1]$, but this neuron returns *pulses* which comprise $y' \in \mathbb{Z} \rightarrow [0, \infty]$. So in the implementation was necessary a transformation of the labels of each class.

The implementation was handled as two dependent modules, a function was implemented that communicates both modules. From the CNN, extract the vector of characteristics obtained from the image and in the opposite direction normalize the cost function (Mean Squared Error mentioned in the Section 4) obtained from the Izhikevich neuron and introduced into the CNN for the backpropagation. The fusion of these modules is seen in Figure 1.

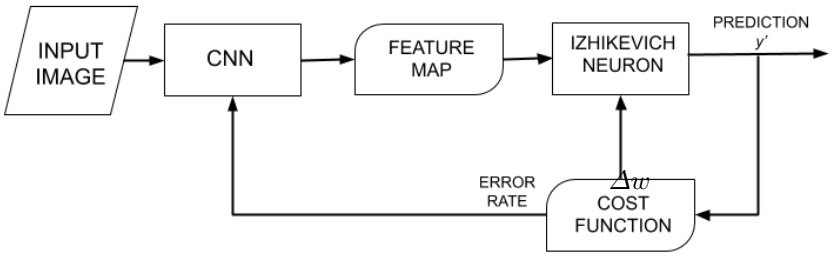


Fig. 1. Network fusion scheme.

3.1 Model Setup

As mentioned earlier, Izhikevich neuron depends on a series of constant parameters. These parameters must be configured in an appropriate way to achieve good classification results, experiments were performed with the configurations proposed by [3] and [15], however the best results were obtained with the configuration proposed in [5]. The parameters used are shown in Table 1.

Table 1. Parameters of the Izhikevich neuron.

Parameter	Value
a	0.03
b	-2
C	100
d	100
v_r	-60
v_t	-40
c	-50
k	1.7
v_{peak}	35
α	1.0^{-5}
w	0.0

Although the original model proposed by Izhikevich establishes the value of the bias θ as a constant, the observation of the antecedent paradigms is that they update this value of bias in a similar way to the weights w, for this reason the experimentation was carried out using a proposed update of θ expressed in

(10), in such a way that it adapts to the data and it is not necessary to do the tuning manually for each case:

$$\theta = \theta + \alpha \cdot (y - y') \cdot (w \times x). \tag{10}$$

The CNN shares some parameters of the Izhikevich neuron as the learning rate α. This proposed model has layers of pooling, in total it has 3 of these, one of max pooling (Equation 4, Section 2.1) with kernel of dimension $N = 3$. The other two consist of layers of average pooling (Equation 5, Section 2.1), of dimensions $N = 4$ and $N = 2$. The justification for this configuration is the reduction of obtained characteristics, since this way a vector of 384 input values for the Izhikevich neuron is achieved. When leaving the configuration without pooling layers, a vector of $588,800$ input values is obtained, which can greatly affect the behavior of the pulses. the whole of this proposed model can be seen in Figure 2.

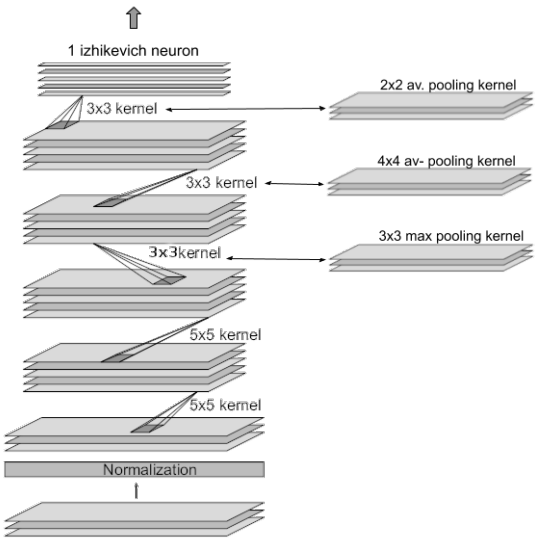


Fig. 2. Proposed model.

As shown in Figure 2, the proposed model is composed of two blocks given by the Convolutional Neuronal Network and Izhikevich neuron. In the CNN block it appears as input an image of size 66×200 with 3 channels (RGB). Layer 1 ($\mathcal{C}1$) convolves the input image with 24 filters of size 5×5 obtaining 24 size feature maps 62×196 without reduction:

$$\mathcal{C}1_{i,j} = \sigma\left(\sum_{d=0}^{2}\sum_{m=0}^{4}\sum_{n=0}^{4} w_{m,n}^{d} \cdot x_{i+m,j+n}^{d} + b\right). \tag{11}$$

To make the output of the nonlinear linear operation, it is advisable that the convolution output is passed through a Linear Units Rectifier (ReLU) $\sigma(x) = \max(0, x)$. Layer 2 ($\mathcal{C}2$) convolves previous maps with 36 filters of size 5×5, obtaining 36 feature maps of 58×192. Layer 3 ($\mathcal{C}3$), the convolution is carried out using 48 filters of 5×5, obtaining 48 maps of 54×188. Data reduction is done from layer 4 ($\mathcal{P}4$) of *Max Pooling* (4) with filter size 3×3 getting the same 48 feature maps but with size 18×62:

$$\mathcal{P}4_{i,j} = \max \begin{pmatrix} \mathcal{C}3_{i,j}\ \mathcal{C}3_{i+1,j} \\ \mathcal{C}3_{i,j+1}\ \mathcal{C}3_{i+1,j+1} \end{pmatrix}. \tag{12}$$

Layer 5 ($\mathcal{C}5$) convolves through 64 filters of 3×3, obtaining 64 maps of 16×60. In layer 6 ($\mathcal{P}6$), the reduction is made through the filter *Average Pooling* (5) of size 4×4, obtaining 64 maps of 4×15:

$$\mathcal{P}6_{i,j} = \frac{\sum_{m=0}^{N-1} \sum_{n=0}^{N-1} \mathcal{C}5_{i+m,j+n}}{N^2}. \tag{13}$$

Layer 7 ($\mathcal{C}7$) convolves with 64 filters of 3×3, obtaining 64 maps of size 2×3. Layer 8 ($\mathcal{P}8$) reduces the size of the maps with a filter *Average Pooling* of 2×2, obtaining 64 maps of 1×6. Finally, layer 9 ($\mathcal{F}$) crushes the maps obtained in P8 of size $d \times m \times n$ in a one-dimensional vector, $\mathcal{F} : \mathcal{P}8^{d \times m \times n} \mapsto \mathcal{P}8^{1 \times (d \times m \times n)}$. Given the output of $\mathcal{F}$, the CNN output vector consists of 384 values.

In the Izhikevich block, x represents the vector $\mathcal{F}$. The x entry is processed by the operations performed in the Algorithm 1, The output of the block consists of the pulse train $y' \in [0, 100]$ given by the activation function (8), similar to a multilayer perceptron in the discrete domain. The pulse trains are interpreted as the discrete prediction obtained from the proposed model, to evaluate the prediction a transformation processes are performed, which is detailed in Section 4.1. The synaptic weights w is a vector of the same length of the inputs x, through the inner product and the control of a hyperparameter θ calculates the synaptic energy of the neuron, it is accumulated during 1000 iterations generating a pulse when it exceeds the threshold v_{peak}. The total of pulses obtained in the neuron is recognized as the prediction, this serves to calculate the loss function which is used for the adjustment of the weights w (9) and retropropagated to the convolutional block.

The convolutional block is adjusted by the optimization algorithm Adam [11], once the Izhikevich block obtains the error function, it is retropropagated to the previous block to do the adjustment of the synaptic weights in the training iteration i (14), where α is the learning rate, (m_i) the exponential moving average of the gradient (loss function), (v_i) is the squared gradient and ϵ a hyperparameter of the algorithm:

$$w_i \leftarrow w_{i-1} - \frac{\alpha_i \cdot m_i}{(\sqrt{v_i} + \epsilon)}. \tag{14}$$

4 Experimental Setup

4.1 Data

As an experimental purpose, it was proposed to work with images captured in the Udacity simulator, which is used for the development of autonomous vehicles. The information provided by this simulator is a pattern of three images obtained from simulated cameras mounted on the vehicle, directed towards the front of the road, the three cameras remain perpendicular to the vehicle and are distributed to the center and one in each respective position of the edges side of the car, these images are shown in Figure 6. Each image has a resolution of 320×160 pixels and is trimmed to remove 60 pixels containing the sky and 25 pixels containing the front of the vehicle. Coupled with this pattern of three images, there are as a class an angle of the steering wheel, which represents the expected action depending on the shape of the road.

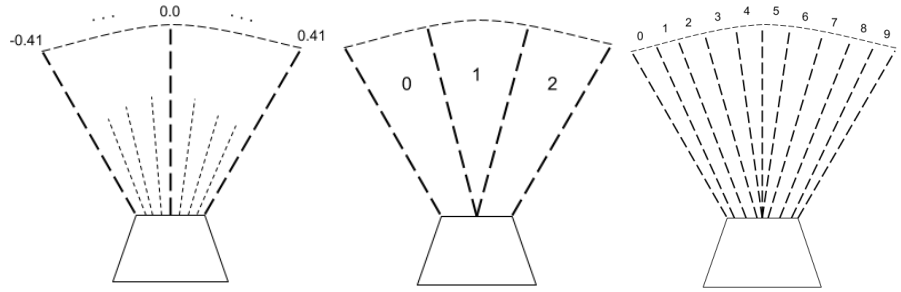
(a). Left camera. **(b).** Center camera. **(c).** Right camera.

Fig. 3. Images obtained from manual driving in the Udacity simulator.

It is worth mentioning that this angle of rotation of the steering wheel is represented in radians and comprises in a closed interval of $y = [-0.41, 0.41]$ so there is a total of 83 classes if only two floats are considered, as shown in Figure 7. A limitation is that this model has only a single classification node, so a problem of 83 classes is an unfair challenge for a single neuron, and on the other hand increasing the number of Izhikevich neurons would represent a computational cost equal to or greater than the Multilayer Perceptron. Therefore, the Equation (16) is proposed to perform the transformation of this angles to N classes. For example in Figure 4b, the angles were divided into 3 intervals conformed by the expressions mentioned in (12). In the same way it can observe the intervals of 10 classes in the Figure 9. This transformation allows experiments to evaluate the performance of the proposed model as the complexity of the problem rises:

$$f(y) = \Big\{ i \text{ if } (y_{min} + \tfrac{|y_{min} - y_{max}|}{classes} i) \leq y < (y_{min} + \tfrac{|y_{min} - y_{max}|}{classes}(i+1)), \quad (15)$$

$$f(y) = \begin{cases} 0 & \text{if } y < -0.13, \\ 1 \text{ if } & -0.13 \leq y < 0.13, \\ 2 & \text{if } 0.13 \leq y. \end{cases} \quad (16)$$



(a). Original data with 83 classes.

(b). Transformed data to 3 classes.

(c). Transformed data to 10 classes.

Fig. 4. Steering wheel angle as class.

However, classification with the Izhikevich neuron returns only positive integers, which are defined as pulses. Therefore, the normalization of the data is proposed using (17) to obtain values in the closed interval $y' = [0, 100]$, in order to compare the output pulses with the real classes of the database:

$$y' = \frac{y - y_{min}}{y_{max} - y_{min}} \cdot 100. \tag{17}$$

Now, if the angles (classes) of the database were scaled with (17) to be represented as pulses, both these angles and the output pulses of the neuron need to be rescaled to the interval $y = [-0.41, 0.41]$ to be later interpreted. To perform this scaling process, is used the inverse algebraic of (17), whose expression is shown in (18):

$$y = \frac{y'}{100} \cdot (y_{max} - y_{min}) + y_{min}. \tag{18}$$

The driving database was created with a total of $11,343$ instances, which contains a total of $34,029$ images. Although the three views are offered, in this experiment it is not the case to predict the exact angle of rotation, for which reason only the image captured by the central camera will be taken into account. For the training parameters, different combinations were made to evaluate the amount of data that can be classified correctly. These experiments involve evaluating both models in training through 50, 100 and 200 epochs to evaluate the learning behavior. The performance test is also carried out in classification, evaluating the classification for 3, 5 and 10 classes, each experiment with their respective validation set equivalent to 20% of the total training lot, using only data that was not used for the training.

4.2 Evaluation

For this experimentation, two types of evaluation are required; The first type is to measure the rate at the proposed model learns from training data and

backpropagation. To perform this measurement it is necessary to graph the values obtained of the cost function through each training epoch. This cost function is given by the Mean Squared Error expressed in (19), the metric obtains the mean of prediction error taking the difference between the expected output and the obtained output. As a support to this metric, it is also proposed to evaluate with the metric of the Root Mean Square Error in (20), observing that one of the classes is negative and the MSE may present variations:

$$MSE = \frac{1}{n}\sum_{i=1}^{n} = (Y_i' - Y_i)^2, \tag{19}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} = (Y_i' - Y_i)^2}. \tag{20}$$

The second type of evaluation measures the quality of the prediction. Therefore, it is proposed to use the metrics derived from the confusion matrix. The main diagonal represents the true positives (TP), that is, the prediction hits. However, to calculate false positives (FP) it is necessary to add the values of the x column except the value belonging to the TP. For the calculation of false negatives (FN) it is the same procedure but now with the sum of the row x. The first metric derived from the confusion matrix is the recall expressed in (21), this metric measures the percentage of patterns that were correctly classified among all the patterns. The second metric is precision, this metric expressed in (22) is similar to the previous one and measures the percentage of success for a single class. Finally, the F measure metric is shown in (23). This method combines precision and recall to obtain a balanced measure:

$$recall = \frac{TP}{TP + FN}, \tag{21}$$

$$precision = \frac{TP}{TP + FP}, \tag{22}$$

$$Fmeasure = 2 \cdot \frac{precision \cdot recall}{precision + recall}. \tag{23}$$

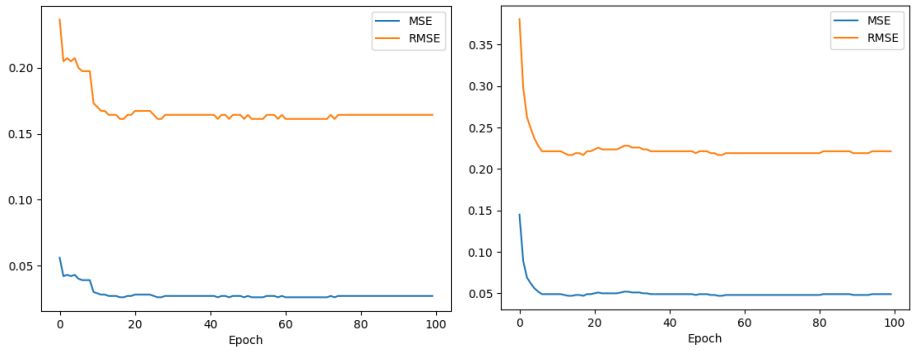
5 Results

The implementation of this fusion was developed in the Python 3.4 language using the Tensorflow 1.1.0 and Keras 2.2 libraries supported by CUDA 8.0 and cuDNN 5.1 drivers for the CNN. The implementation of the Izhikevich neuron was done in Python t0o without the support of the mentioned libraries. The environment used for this experimentation consists of an Intel Core 17-7700 with 4 cores 8 threads 3.6 GHz, SSD storage 120 GB and a GPU Titan X, 3072 cores CUDA 1075 MHz 12 GB RAM. All under the Xubuntu 14.04 Trusty Thar operating system.

In order to compare the experimentation of the proposed model, the deep neural model Pilotnet [1,2] was evaluated. This model was designed to train with the 83 classes mentioned in Section 4.1, but it was evaluated in the same way as the proposed model.

5.1 Training Results

As mentioned in the Section 4.2, the experimentation was carried out under the criteria of training and classification quality. For this first experiment, the MSE and RMSE cost metrics were plotted to measure the rhythm of learning of the proposed model. In short, the graph should show a slope that tends to 0, it indicates that a better classification is being done in the validation stages after a training period. In Figure 13 shows the comparison of the gradients obtained in the training of 100 epochs.



(a). Training with 100 epochs and 5 classes.

(b). Training with 100 epochs and 3 classes.

Fig. 5. Mean Square Error and Mean Quadratic Error obtained during the training of the proposed model.

The gradients obtained in the previous experiment are acceptable since they trend to 0. However, by zooming in on the graph, an erratic behavior in the rhythm of learning is appreciated. These variations depend to a large extent on the dispersion and balance of the classes. Although the gradient is maintained or grows, it remains in acceptable values to make a good classification. The erratic behavior mentioned above can be seen in Figure 14, where a comparison is also made with the gradient of the Pilotnet model.

Observing the graph of the gradient, a great difference can be seen between the learning of both neuronal models. On the other hand, it is appreciated that the proposed model reaches a minimum learning point in which despite adding training epochs this does not improve the results. This observation does not represent a disadvantage, this is an advantage to demonstrate that this neuronal

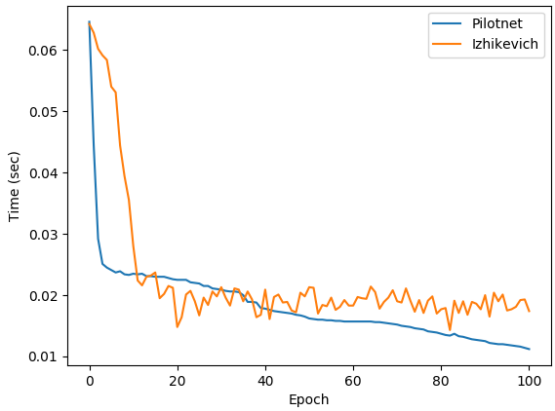


Fig. 6. Cost function in training with 100 epochs and 10 classes.

paradigm requires fewer training periods, however it is limited by capacity and data. Now, the training time is clearly shorter in the proposed model. The time that take to train in the experiments are shown in Figure 15.

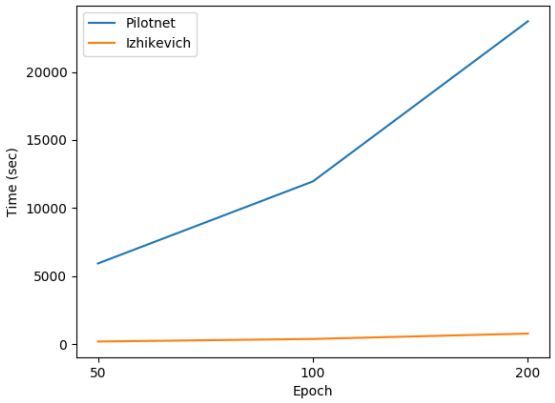


Fig. 7. Time of training for both models.

The time graph shows a big difference in the time it takes to train both models. This is due to the reduction of complexity with respect to the comparison model. To better visualize the time obtained in the experiments, Table 2 is shown, where the time is in seconds for each experiment.

Table 2. Time that took the experiments.

Model	Training Time (sec)
Proposal 50 epochs	189.19
Pilotnet 50 epochs	5,925.55
Proposal 100 epochs	378.53
Pilotnet 100 epochs	11,954.79
Proposal 200 epochs	774.40
Pilotnet 200 epochs	23,719.89

5.2 Classification Results and Comparison

Given the reference, the experimentation was carried out equitably with 100 epochs of training for both models. This evaluation is done for 3, 5 and 10 classes with the intention of evaluating the maximum capacity of the Izhikevich neuron for the classification. The results obtained are shown in Table 3.

Table 3. Results obtained for both models in classification with 100 epochs of training.

Model	Metrics		
	Precision	Recall	F-Measure
Proposal 3 classes	0.991	0.983	0.987
Pilotnet 3 classes	0.999	0.994	0.996
Proposal 5 classes	0.855	0.845	0.849
Pilotnet 5 classes	0.991	0.966	0.966
Proposal 10 classes	0.740	0.741	0.740
Pilotnet 10 classes	0.911	0.897	0.903

Although the model that classifies through the multilayer perceptron presents slightly results, this considerably increases the training time. The proposed model requires much less time of the referent model, even though the performance results in classification are very close in case of 3 classes. Although in the case of increasing the number of classes reduce its classification capacity, depending on the problem to be treated, it could be considered as a faster option. Recalling that this generation of neural networks are in research for new applications.

Another aspect to consider is that the Izhikevich neuron depends on a series of constant values. Although for most of these constants there is a recommended value, in the specific case of the threshold (θ) it is observed that for most of the neuronal paradigms this parameter must be adjustable as the weights (w). As a proposal to contribute to the development of this neuronal paradigm, an update of the threshold is proposed given the function expressed in (10). It is worth mentioning that the experimentation reported here was implemented considering the aforementioned update and showing good results.

6 Conclusions and Future Improvements

As demonstrated in this research, one way to reduce the training time of a Neural Network is to use an alternative to the Multilayer Perceptron Network commonly used for classification. This simple but effective solution is able to reduce training time considerably due to the remarkable decrease in neurons to be processed. As presented in this article, the Izhikevich neuron is able to correctly classify the patterns obtaining metrics very close to those of the reference model in a problem of medium complexity with 3 classes. Although in the literature is reported that this type of neuron turns out to be useful for binary nonlinear classification, thanks to the adjustment of the Convolutional Neuronal Network it can improve the results allowing classifying with more classes.

Although the experimentation shown here consists of not extensive training sets with the full number of classes, the improvement of this proposed model will be able to result in problems of classification in Big Data, which is planned as a future work. Similarly, the dynamic adjustment of the parameters of the Izhikevich neuron, such as the one shown in the Section 3.1 Equation (10), is an aspect to be improved since it depends to a large extent on experiments prior, and as is well known, a neural network is a non-parametric algorithm.

Acknowledgment. The authors express their gratitude to CONACYT, as well as Tecnológico Nacional de México/CENIDET for the support given to the department of computational sciences, in which it was possible to carry out this investigation that is part of the project 5628.19-P so-called "Sistema embebido para asistencia de conducción basado en Lógica Difusa Tipo-2".

References

1. Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., Zhang, J., et al.: End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316 (2016)
2. Bojarski, M., Yeres, P., Choromanska, A., Choromanski, K., Firner, B., Jackel, L., Muller, U.: Explaining how a deep neural network trained with end-to-end learning steers a car. arXiv preprint arXiv:1704.07911 (2017)
3. Demirkol, A.S., Ozoguz, S.: A low power vlsi implementation of the izhikevich neuron model. In: 2011 IEEE 9th International New Circuits and systems conference. pp. 169–172. IEEE (2011)
4. van Grinsven, M.J., van Ginneken, B., Hoyng, C.B., Theelen, T., Sánchez, C.I.: Fast convolutional neural network training using selective data sampling: Application to hemorrhage detection in color fundus images. IEEE transactions on medical imaging 35(5), 1273–1284 (2016)
5. Hernández-Becerra, M.M.L.: Clasificación de patrones mediante el uso de una red neuronal pulsante. Congreso Mexicano de Inteligencia Artificial 5 (2016)
6. Iakymchuk, T., Rosado-Muñoz, A., Guerrero-Martínez, J.F., Bataller-Mompeán, M., Francés-Víllora, J.V.: Simplified spiking neural network architecture and stdp learning algorithm applied to image classification. EURASIP Journal on Image and Video Processing 2015(1), 4 (2015)

7. Iandola, F.N., Moskewicz, M.W., Ashraf, K., Keutzer, K.: Firecaffe: near-linear acceleration of deep neural network training on compute clusters. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2592–2600 (2016)
8. Izhikevich, E.M.: Simple model of spiking neurons. IEEE Transactions on neural networks 14(6), 1569–1572 (2003)
9. Izhikevich, E.M.: Which model to use for cortical spiking neurons? IEEE transactions on neural networks 15(5), 1063–1070 (2004)
10. Jerry, M., Chen, P.Y., Zhang, J., Sharma, P., Ni, K., Yu, S., Datta, S.: Ferroelectric fet analog synapse for acceleration of deep neural network training. In: 2017 IEEE International Electron Devices Meeting (IEDM). pp. 6–2. IEEE (2017)
11. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
12. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. nature 521(7553), 436 (2015)
13. Matsubara, T., Torikai, H., Hishiki, T.: A generalized rotate-and-fire digital spiking neuron model and its on-fpga learning. IEEE Transactions on Circuits and Systems II: Express Briefs 58(10), 677–681 (2011)
14. Pattanayak, S.: Pro Deep Learning with TensorFlow. Springer (2016)
15. Rice, K.L., Bhuiyan, M.A., Taha, T.M., Vutsinas, C.N., Smith, M.C.: Fpga implementation of izhikevich spiking neural networks for character recognition. In: 2009 International Conference on Reconfigurable Computing and FPGAs. pp. 451–456. IEEE (2009)
16. Scherer, D., Müller, A., Behnke, S.: Evaluation of pooling operations in convolutional architectures for object recognition. In: Artificial Neural Networks–ICANN 2010, pp. 92–101. Springer (2010)
17. Smith, L.N., Topin, N.: Super-convergence: Very fast training of neural networks using large learning rates. arXiv preprint arXiv:1708.07120 (2017)
18. Zhang, J.R., Zhang, J., Lok, T.M., Lyu, M.R.: A hybrid particle swarm optimization–back-propagation algorithm for feedforward neural network training. Applied mathematics and computation 185(2), 1026–1037 (2007)