Engineering and Technology Journal e-ISSN: 2456-3358
Volume 07 Issue 07 July-2022, Page No.-1369-1375
DOI: 10.47191/etj/v7i7.07, I.F. – 6.484

A Deep-Fracture Approach to Embedded Multilayer Neural Networks for Handwritten Character Recognition

Fredy Martínez[1], Angélica Rendón[2]
[1,2]Facultad Tecnológica, Universidad Distrital Francisco José de Caldas, Bogotá, Colombia

ABSTRACT: Low-cost microcontrollers, but also with reduced processing and memory capabilities, are highly appreciated devices for the development of specific tasks as embedded systems. The devices found in the market today provide dedicated 32-bit architectures with communication capabilities ideal for IoT applications. Some of these tasks would highly benefit from the generalization and approximation capabilities of a neural network, but current convolutional networks prove to be too complex for microcontrollers. An alternative for some cases is the use of Multilayer Neural Networks (MNNs), which, thanks to a shallower depth, reach sizes suitable for use in small embedded systems. An MNN can accommodate a complex decision surface capable of being used in high dimensionality pattern classification problems, such as handwritten character recognition. Given the lack of an explicit strategy to define the architecture of these networks in coherence with the problem to be solved, this paper conducts a performance analysis of these networks in terms of their depth, to establish criteria to determine their size in a specific application. For the evaluation, the public database MNIST has been used in conjunction with classical metrics to evaluate the performance of a classification model. The results show that the depth of the network determines to a high degree the performance of the model, and guides the appropriate selection of the architecture.

KEYWORDS: Accuracy, Confusion matrix, Gradient-based learning, Machine learning, Multilayer neural network, Optical character recognition.

I. INTRODUCTION

Although the automotive sector is the one that gives greater visibility to embedded computing systems, the truth is that these systems are increasingly present in everyday life, permeating areas as important to humans as wearable systems for health monitoring, control and sensing in military applications, robotics (with particular relevance in service robotics), applications supported by the internet of things (IoT), and even in household appliances and technological toys (Castañeda & Salguero, 2017, Zilong et al., 2018, Martínez et al., 2019, Gajaria & Adegbija, 2022). These embedded systems are supported on a wide variety of hardware, ranging from FPGAs (Field-Programmable Gate Arrays) and small microcontrollers from 16 to 32 bits, to development boards with a memory management unit that allow running graphical operating systems such as Linux (Moreno & Páez, 2017, Esquivel et al., 2012). In general, these systems are part of a larger system, and require custom-developed software due to hardware resource constraints (Ktari et al., 2022). Even so, they have great advantages in terms of performance and low cost, which is why they have become so popular. Another important aspect of their current popularity is their ability to incorporate automatic learning strategies, which together with their Wi-Fi and Bluetooth communication capabilities make them powerful processing systems (Hashemifar et al., 2019).

Deep networks have turned around the performance of systems in tasks such as speech recognition, computer vision, and language processing, applications in which traditional filtering-based strategies perform poorly (Caley et al., 2019, Martínez et al., 2018a). These learning techniques, in conjunction with reinforcement learning strategies, are crucial in pattern recognition applications (Hock & Schoelling, 2019, Martínez & Rendón, 2022). Thanks to the possibility of training and evaluating deep networks on current computers, it has been possible to implement such activities as modules in key areas, such as robotics, and in particular, assistive care, improving integration with humans, while simultaneously increasing the capacity of the systems (Yarza et al., 2022). Major drivers have been both the reduced cost of manufacturing high-performance processors, as well as the lower cost and greater capacity for storing information in online servers. However, this type of solution is still relegated to next-generation embedded systems because the hardware they use does not yet reach the required performance levels (Prabakaran et al., 2022). This resource limitation still allows the implementation of some less demanding machine learning models, such as Multilayer

Neural Networks (MNN), with a much shallower architecture, and with very high performance in some difficult applications for traditional digital algorithms (Hu et al., 2021).

Machine learning strategies such as SVM (Support Vector Machine) and CNN (Convolutional Neural Network) are highly resources demanding both in their training process and in their propagation process (more in the former than in the latter, but even an off-line system is demanding in high speed and storage) (Martínez et al., 2018b). Today's computers have optimized processors with pipelining, deep cache memory hierarchies, and multicore units, which can meet the demands of these models, especially if they work hand in hand with graphics processing units (GPU) (Yao et al., 2021). In this sense, there is research to develop similar acceleration structures for FPGAs, but to date, embedded systems do not have these features (Riaño et al., 2012, Xu et al., 2022). This makes research on optimization schemes for machine learning techniques as well as hardware architectures a very active field of research. It is also worth mentioning that classical connectionist schemes such as MNNs have been widely reported in the literature since the 1980s as a suitable strategy for solving some problems, such as handwritten character recognition (Shi et al., 2022, Ao et al., 2022).

This research examines one of the key issues in the design of MNN applications on an embedded system, which is related to the design of the model architecture according to the expected performance. A good (high performance) pattern recognition system can be built on the limited hardware of a microcontroller if special care is taken on system parameters such as training information, training information normalization, learning scheme, and above all, classification model architecture. MNNs have been reported to be successful in automatic character recognition applications, but this type of network does not have a rigid structure related to the problem conditions, which makes its design complex, and becomes a trial-and-error problem. In the search for a suitable architecture for this problem, we propose a performance evaluation of different MNN models by varying step by step the depth of its structure and evaluating in each case the performance achieved. We use as a case study the problem of character recognition supported by the well-known MNIST database. From the results of the research, we show how the network architecture is fundamental in the process of identifying handwriting features, and achieving the definition of an optimal depth for a 32-bit microcontroller.

The following part of the paper is arranged in this way. Section II presents the formulation of the problem, details and design considerations, preliminary concepts, background work of the research group and the development methodology adopted for the solution of the problem following the specifications of the problem profile. Section III shows in detail the results found during the research. We present the data corresponding to the performance analysis, and we discuss how the performance of the scheme is affected under each case. And finally, in Section IV, we present our conclusions.

II. MATERIALS AND METHODS

To evaluate the performance of MNN as a function of its depth, the MNIST database was used. This corresponds to a large database of handwritten digits that is commonly used to train various image processing systems (Fig. I). It has a training set of 60000 samples and a test set of 10000 samples. It is a subset of a larger set available from NIST. The digits have been size normalized and centered on a fixed size image of 28x28 pixels. This database can be loaded directly into the Keras framework in which all the code was developed. We use Keras 2.8.0 with TensorFlow 2.8.2 and Python 3.10.5.

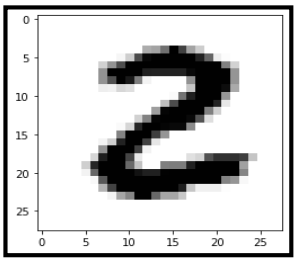

Figure I. Sample from the MNIST Public Database

The structure of the training and validation data was maintained throughout the training with all models, but in each case, the images were previously randomly shuffled to avoid bias. We used five MNN models characterized by different depths, one model with one hidden layer, one with two hidden layers, one with three hidden layers, one with four hidden layers, and one with five hidden layers. All layers were designed in the same way, keeping 100 neurons in each layer, and 10 output categories. All layers used the sigmoid activation function and softmax activation function in the output layer. The models were built with the Keras Sequential class, adding the necessary hidden layers as appropriate. The input vector of each model was formed with the concatenated and normalized rows of the images, which constituted a row vector (1D tensor) of 784 elements. Figs. II to VI shows the detailed structure of each of the five models evaluated.

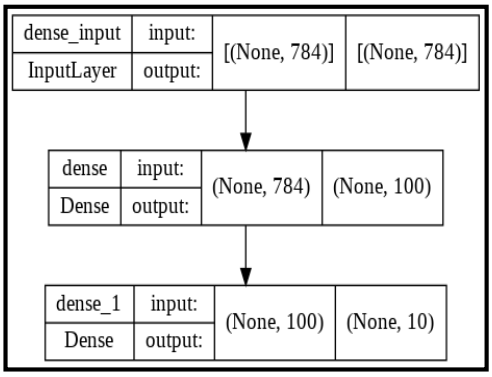

Figure II. MNN Architecture with a Single Hidden Layer

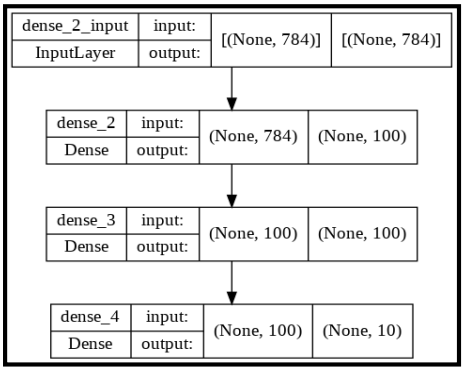

Figure III. MNN Architecture with Two Hidden Layers

According to the detailed structure, the model with one hidden layer requires 79510 parameters, the model with two hidden layers requires 89610 parameters, the model with three hidden layers requires 99710 parameters, the model with four hidden layers requires 109810 parameters, and the model with five hidden layers requires 119910 parameters. These parameters are what will consume the storage resources of the microcontroller, so the ideal choice is the model with the highest performance and the lowest number of parameters. All parameters of these models must be tuned by training.

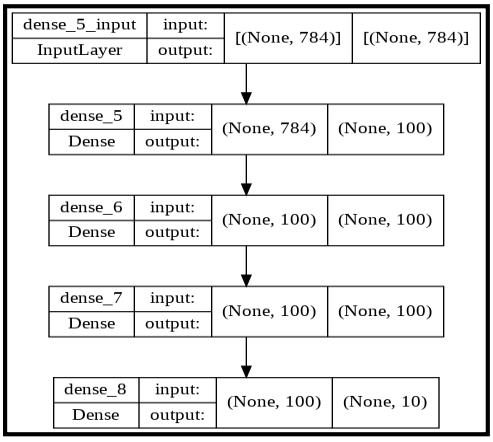

Figure IV. MNN Architecture with Three Hidden Layers

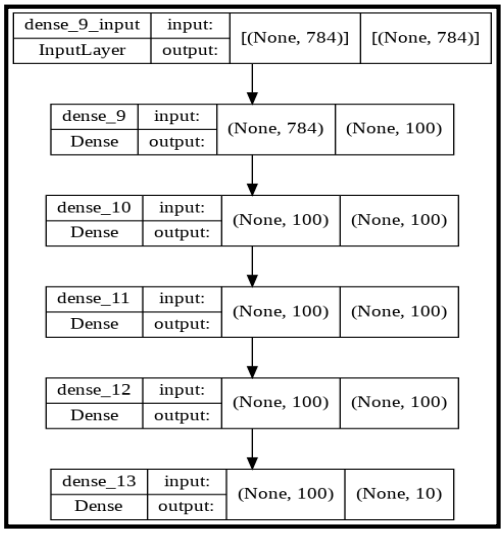

Figure V. MNN Architecture with Four Hidden Layers

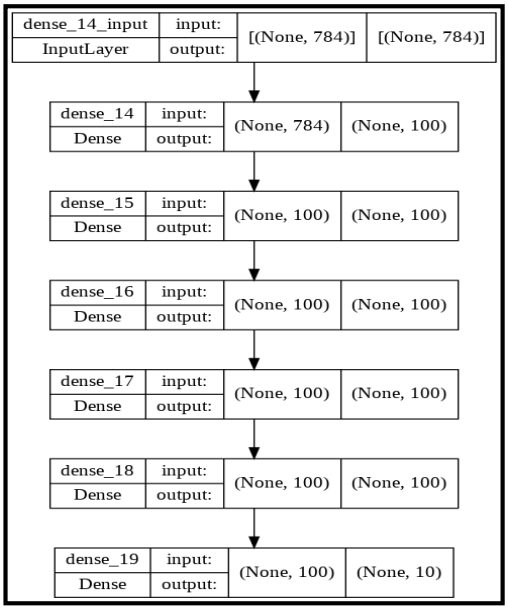

Figure VI. MNN Architecture with Five Hidden Layers

All models were trained and evaluated under the same conditions. For training, the categorical cross-entropy function was used as the loss function, and Stochastic Gradient Descent (SGD) was used as the optimizer. The training was performed over 30 epochs with a batch size of 100. In each case, a set of random parameters generated with different seeds dependent on several variables, including the machine clock, was started. For evaluation, all models have evaluated over 30 epochs with both training and validation data. The metrics used were accuracy and Mean Squared Error (MSE). Also for each trained model, the confusion matrix was calculated for the validation data (10000 data support).

III. RESULTS AND DISCUSSION

As mentioned, the code was developed in Python with Keras support. An Intel Core i7-7700HQ eight-core machine was used with a Debian 11 Bulleye Linux OS with kernel 5.18. Google Colab support was available, configuring a GPU in its runtime. Each epoch on each model consumed a little more than two seconds, for average training times of 1 minute. This time could be influenced by the data visualization, as this visualization was requested for error and metrics in each training epoch, both for training data and validation data. In principle, the training time is the same for all five models. The five models were stored in separate objects in the same training environment and examined simultaneously.

Fig. VII shows the loss function behavior achieved by each of the five models. Fig. VII(a) shows the curves for the MNN model with one hidden layer, Fig. VII(b) the curves for the model with two hidden layers, Fig. VII(c) the curves for the model with three hidden layers, Fig. VII(d) the curves for the model with four hidden layers, and Fig. VII(e) the curves for the model with five hidden layers. In these figures, the blue curve corresponds to the training data, while the red dashed curve corresponds to the unknown validation data. The first

interesting behavior in each of the five curves in this figure is that the validation data have similar behavior to the training data, i.e., the behavior of the loss function is similar in both cases. This means that initially none of the models suffers from overfitting. The model with one hidden layer has the best behavior, the error is consistently reduced throughout the whole process, and it could even be considered to train over a larger number of epochs since no overfitting is observed. The model with two hidden layers also performs well, and the error, although higher than the first case, is not so far behind, and again it is possible to think of training the model over a larger number of epochs since no overfitting is observed. The model with three hidden layers is inferior to the first two cases, the error is reduced consistently throughout the training but much more slowly at the beginning of the training. From epoch 10 onwards the error reduces with a steep slope similar to that observed in the training of the first model, and no saturation or overfitting is observed so that it can be contemplated to train this model for a larger number of epochs. However, under these conditions, the model would require more resources and time to reach a competitive structure with the first two cases. The model with four hidden layers showed very little reduction in training error, and with the data collected it is not possible to infer whether the performance will improve for a larger number of epochs. The same, but with worse results are observed with the last model since in this case no error reduction was observed throughout the training process. These results should be consistent with the other metrics but indicate that a greater degree of network depth considerably deteriorates its ability to retain the model's feature information.

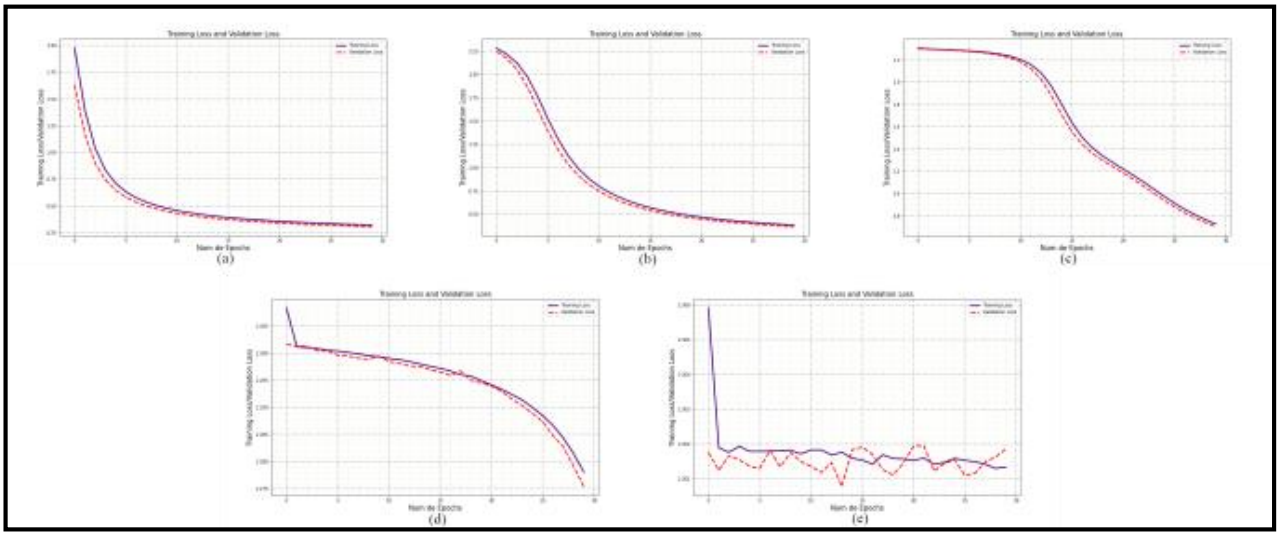

Figure VII. Behavior of the Loss Function During Training. The Blue Line Corresponds to the Training Data and the Red Line to the Validation Data. (A) One Hidden Layer, (B) Two Hidden Layers, (C) Three Hidden Layers, (D) Four Hidden Layers, and (E) Five Hidden Layers

Fig. VIII shows the behavior of the accuracy for the five models throughout the training, again the blue curve corresponds to the training data, while the red dotted curve corresponds to the unknown validation data. Fig. VIII(a) corresponds to the model with a single hidden layer, Fig. VIII(b) corresponds to the model with two hidden layers, Fig. VIII(c) corresponds to the model with three hidden layers, Fig. VIII(d) corresponds to the model with four hidden layers, and Fig. VIII(e) corresponds to the model with five hidden layers. As expected, this metric has results congruent with those shown for the loss function in Fig. VII. In none of the models, traces of overfitting are observed given the closeness between curves corresponding to training and validation. The first two models (with one and two hidden layers) seem to have reached saturation in the parameter fitting process, and their accuracy exceeds or is close to 90%. When the model incorporates a single hidden layer the accuracy increases rapidly in the first 10 epochs, then its growth is slower but continuous. When the model incorporates two hidden layers, the growth of the accuracy is slower, requiring up to 20 epochs to increase its value considerably. These first two models have high performance and are suitable for embedded implementation, but the second model requires a little more than 10000 additional parameters to the model with a single hidden layer. In the case of the model with three hidden layers, under the same conditions, its accuracy does not exceed 80%, which is not a bad result, and the curves show that the model can continue to learn over a larger number of epochs. The last two models (four and five hidden layers) do not exceed 20% accuracy, and in both cases, it is difficult to establish any success in their training.

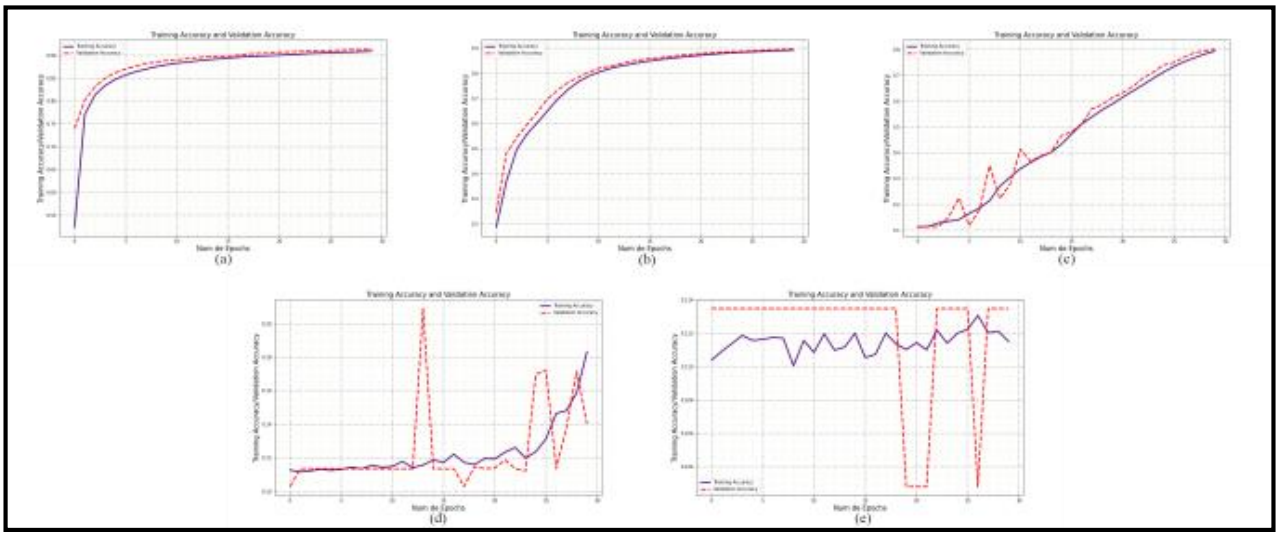

Figure VIII. Behavior of the Accuracy during Training. The Blue Line Corresponds to the Training Data and the Red Line to the Validation Data. (A) One Hidden Layer, (B) Two Hidden Layers, (C) Three Hidden Layers, (D) Four Hidden Layers, And (E) Five Hidden Layers

The last metric calculated for the models was the confusion matrix, which is shown in Fig. IX. Fig. IX (a) corresponds to the matrix for the one-hidden-layer model, Fig. IX (b) corresponds to the matrix for the two-hidden-layer model, Fig. IX(c) corresponds to the matrix for the three-hidden-layer model, Fig. IX (d) corresponds to the matrix for the four-hidden-layer model, and Fig. IX (e) corresponds to the matrix for the five-hidden-layer model. The confusion matrix of the models was calculated only for the validation data, which constitutes support of 10000 images. The categories predicted by the models were placed in the rows and their correct assignments in the columns. Given the amount of data, to facilitate their analysis a color temperature was assigned to the values, which is shown on the right of each sub-figure. In this coding, darker colors indicate a low number of elements, while lighter colors indicate a high number of elements (around 1000 elements per category are expected). Under these conditions, it is easy to observe that the first three models manage to correctly categorize the elements in their categories (one, two, and three hidden layers), while the last two models do not (four and five hidden layers). Furthermore, it is observed that the performance of the first two models exceeds the requirements for the construction of the embedded system.

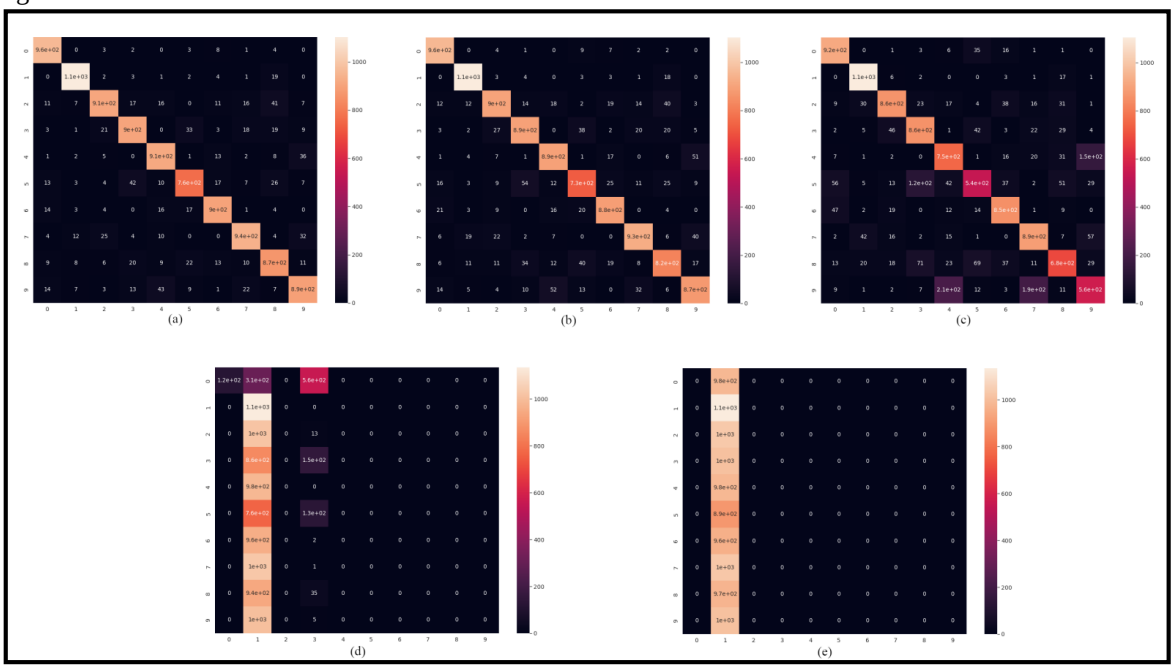

Figure IX. Behavior of the Confusion Matrix for Validation Data: (A) One Hidden Layer, (B) Two Hidden Layers, (C) Three Hidden Layers, (D) Four Hidden Layers, And (E) Five Hidden Layers

IV. CONCLUSION

In this paper, we evaluate the performance of an MNN model for its possible use on embedded systems, particularly on small 32-bit microcontrollers. This evaluation consisted in determining the performance of the model against different network depths. Fixed-size of 100 neurons per layer and the same training and evaluation conditions were selected. MNIST was used as the dataset since the final application contemplates the autonomous identification of handwritten characters. The training was developed in all cases over 30 epochs with cross-entropy function as loss function and SDG as optimizer. As metrics for performance evaluation, the accuracy in each training epoch was calculated with the training and validation data, and the confusion matrix with the validation data. The final results show that, for a short training of 30 epochs, a higher network depth deteriorates the learning capability of the models when using a vanilla SDG as an optimizer. It is possible that longer training cycles would achieve better performance for deeper networks, but an accuracy above 90% is achieved with a hidden layer, a model that also proposes the lowest RAM consumption. Future research developments consider a more complex database (larger categories including the entire Latin alphabet), as well as an evaluation of the behavior of the models against a much larger number of training cycles.

ACKNOWLEDGMENT

This work was supported by the Universidad Distrital Francisco José de Caldas, in part through CIDC, and partly by the Facultad Tecnológica. The views expressed in this paper are not necessarily endorsed by Universidad Distrital. The authors thank the research group ARMOS for the evaluation carried out on prototypes of ideas and strategies.

REFERENCES

1. Ao, X., Zhang, X.Y., & Liu, C.L., "Cross-modal prototype learning for zero-shot handwritten character recognition", *Pattern Recognition*, 2022, 131(1):108859, ISSN 0031-3203, doi:10.1016/j.patcog.2022.108859.
2. Caley, J., Lawrance, N., & Hollinger, G., "Deep learning of structured environments for robot search", *Autonomous Robots*, 2019, 43(7):1695–1714, ISSN 0929-5593, doi:https://doi.org/10.1007/s10514-018-09821-4.
3. Castañeda, J. & Salguero, Y., "Adjustment of visual identification algorithm for use in stand-alone robot navigation applications", *Tekhnê*, 2017, 14(1):73–86, ISSN 1692-8407.
4. Esquivel, J., Marín, N., & Martínez, F., "Digital development platform based on game boy advance and the ARM7 architecture", *Tekhnê*, 2012, 9(1):5–12, ISSN 1692-8407.
5. Gajaria, D. & Adegbija, T., "Evaluating the performance and energy of STT-RAM caches for real-world wearable workloads", *Future Generation Computer Systems*, 2022, 136(1):231–240, ISSN 0167-739X, doi:10.1016/j.future.2022.05.023.
6. Hashemifar, Z., Adhivarahan, C., Balakrishnan, A., & Dantu, K., "Augmenting visual slam with wi-fi sensing for indoor applications", *Autonomous Robots*, 2019, 43(8):2245–2260,ISSN 0929-5593, doi:https://doi.org/10.1007/s10514-019-09874-z.
7. Hock, A. & Schoelling, A., "Distributed iterative learning control for multi-agent systems", *Autonomous Robots*, 2019, 43(8):1989–2010, ISSN 0929-5593, doi:https://doi.org/10.1007/s10514-019-09845-4.
8. Hu, J., Wang, P., Xu, C., Zhou, H., & Yao, J., "High accuracy adaptive motion control for a robotic manipulator with model uncertainties based on multilayer neural network", *Asian Journal of Control*, 2021, 24(3):1503–1514, ISSN 1561-8625, doi:10.1002/asjc.2546.
9. Ktari, J., Frikha, T., Hamdi, M., Elmannai, H., & Hmam, H., "Lightweight AI framework for industry 4.0 case study: Water meter recognition", *Big Data and Cognitive Computing*, 2022, 6(3):72, ISSN 2504-2289, doi:10.3390/bdcc6030072.
10. Martínez, F., Jacinto, E., & Montiel, H., "Neuronal environmental pattern recognizer: Optical-by-distance LSTM model for recognition of navigation patterns in unknown environments", *Communications in Computer and Information Science*, 2019, 1071(1):220–227, ISSN 1865-0929,doi:https://doi.org/10.1007/978-981-32-9563-6_23.
11. Martínez, F., Penagos, C., & Pacheco, L., "Deep regression models for local interaction in multi-agent robot tasks", *Lecture Notes in Computer Science*, 2018a, 10942(1):66–73, ISSN 0302-9743, doi:10.1007/978-3-319-93818-9_7.
12. Martínez, F. & Rendón, A., "Implementation and evaluation of advantage actor-critic algorithm on a desktop computer with a multi-core CPU", *International Journal of Computer Applications Technology and Research*, 2022, 11(07):284–290, ISSN 2319-8656, doi:10.7753/ijcatr1107.1005.
13. Martínez, F., Rendón, A., & Arbulú, M., "A data-driven path planner for small autonomous robots using deep regression models", *Lecture Notes in Computer Science*, 2018b, 10943(1):596–603, ISSN 0302-9743, doi:https://doi.org/10.1007/978-3-319-93803-5_56.
14. Moreno, A. & Páez, D., "Performance evaluation of ros on the raspberry pi platform as os for small

robots", *Tekhnê*, 2017, 14(1):61–72, ISSN 1692-8407.

15. Prabakaran, G., Vaithiyanathan, D., & Ganesan, M., "FPGA based intelligent embedded system for predicting the productivity using fuzzy logic", *Sustainable Computing: Informatics and Systems*, 2022, 35(1):100749, ISSN 2210-5379, doi:10.1016/j.suscom.2022.100749.
16. Riaño, J., Ladino, C., & Martínez, F., "Implementation of fft transform on a FPGA oriented towards its application in power electronic converters", *Tekhnê*, 2012, 9(1):21–32, ISSN 1692-8407.
17. Shi, W., Huang, Z., Huang, H., Hu, C., Chen, M., Yang, S., & Chen, H., "LOEN: Lensless opto-electronic neural network empowered machine vision", *Light: Science & Applications*, 2022, 11(1):121, ISSN 2095-5545, doi:10.1038/s41377-022-00809-5.
18. Xu, F., Guo, Z., Chen, H., Ji, D., & Qu, T., "A custom parallel hardware architecture of nonlinear model-predictive control on FPGA", *IEEE Transactions on Industrial Electronics*, 2022, 69(11):11569–11579, ISSN 0278-0046, doi:10.1109/tie.2021.3118427.
19. Yao, Y., Liu, S., Wu, S., Ni, J., Wang, J., Yang, G., & Zhang, Y., "WAMP22S: workload-aware GPU performance model based pseudo-preemptive real-time scheduling for the airborne embedded system", *IEEE Transactions on Parallel and Distributed Systems*, 2021, 33(11):2767–2780, ISSN 1045-9219, doi:10.1109/tpds.2021.3134269.
20. Yarza, I., Agirre, I., Mugarza, I., & Cerrolaza, J.P., "Safety and security collaborative analysis framework for high-performance embedded computing devices", *Microprocessors and Microsystems*, 2022, 93(1):104572, ISSN 0141-9331, doi:10.1016/j.micpro.2022.104572.
21. Zilong, H., Jinshan, T., Ziming, W., Kai, Z., Ling, Z., & Qingling, S., "Deep learning for image-based cancer detection and diagnosis - a survey", *Pattern Recognition*, 2018, 83(1):134–149, ISSN 0031-3203, doi:https://doi.org/10.1016/j.patcog.2018.05.014.