International Journal of Combinatorial Optimization Problems and Informatics, 11(1), Jan-April 2020, 97-105. ISSN: 2007-1558.

www.editada.org

A Convolutional Neural Network for Handwritten Digit Recognition

María Cristina Guevara Neri[1], Osslan Osiris Vergara Villegas[2], Vianey Guadalupe Cruz Sánchez[2], Manuel Nandayapa[2], and Juan Humberto Sossa Azuela[3]

[1] Universidad Autónoma de Ciudad Juárez, Instituto de Ingeniería y Tecnología, Doctorado en Ciencias en Ingeniería (DOCIA).
[2] Universidad Autónoma de Ciudad Juárez, Instituto de Ingeniería y Tecnología.
[3] Instituto Politécnico Nacional (CIC-IPN)
Al171517@alumnos.uacj.mx, {overgara, vianey.cruz, mnandaya}@uacj.mx, humbertososssa@gmail.com

Abstract. Technological development in recent years has generated the constant need to digitalize and analyze data, where handwritten digit recognition is a popular problem. This paper focuses on the creation of two handwritten digit datasets and their use to train a Convolutional Neural Network (CNN) to classify them, also, a proposed extra preprocessing technique is applied to the images of one of the data sets. Experiments show that the proposed preprocessing technique lead to obtain accuracies above 98%, which were higher than the values obtained with the dataset without the additional preprocessing.

Keywords: CNN, Digit Recognition, Image Preprocessing.

Article Info
Received Dec 26, 2016
Accepted Dec 11, 2017

1 Introduction

Handwritten character recognition is still, nowadays, a difficult challenge to face given the existence of many variations of handwriting style [1]. There is an extensive research on the handwritten character recognition (or classification) problem, on all type of handwritten characters from arabic [2], english latin [3], numbers [4], [5] or overall mathematical handwriting [6], [7], [8].

There are several classification techniques for image recognition, such as approximate subgraph matching and levenshtein distance [9], directional feature based on Arnold transform [10], *k*-Nearest-Neighbor (*k*-NN) [11], [12], support vector machine (SVM) [13], [14], convolutional neural networks (CNN) [15], [16] and even combinations of various techniques [17], [18].

The development of an artificial system that classifies successfully any type of handwritten character is not a simple task, even though for a human being it is. A Convolutional Neural Networks (CNN) is a system introduced in 1990 by Yann LeCun *et al* [19], [20], that is proven to be a good performer on image classification problems [21].

A CNN is a neural network with multiple layers, trained with a version of the back-propagation algorithm and it is designed to recognize visual patterns directly from pixel images with minimal preprocessing. Many network architectures are used nowadays, such as AlexNet [21], LeNet-5 [22], GoogLeNet [23] and many more. Where it is worth noting that LeNet-5 is a popular architecture used in handwritten digit recognition problems, given that it was created to classify the Modified National Institute of Standards and Technology (MNIST) database [24], which is a large database of handwritten digits.

There are many image databases used to train a CNN, and the selection of the database depends on what type of character is going to be classified. The MNIST [25] and CIFAR-100 [26] databases, for example, are two of the most common used databases in handwritten recognition problems, with the MNIST database having 28x28 grayscale images with 60,000 training samples and 10,000 test samples and 10 classes, and the CIFAR-100 database having 32x32 RGB images, 50,000 training images and 10,000 test images and 100 classes.

In this paper, two handwritten digits data sets different from MNIST for testing classification algorithms are presented. Then, the LeNet-5 model was trained with both data sets. At the end, it is demonstrated that by applying basic image preprocessing techniques to one data set helps increasing the recognition accuracy compared with the non-preprocessed data set.

The rest of the paper is organized as follows. Section 2 presents the materials and methods used in the experiments, where the creation of the datasets is detailed, as well as the model of the CNN implemented. Then, the experiments and the experimental results are described and discussed in section 3. Finally, section 4 show the conclusions of the research.

2 Materials and Methods

In this section, the set-up where the experiments were performed is discussed. Then, the dataset used in the training and validation in the CNN implemented to solve the classification problem is described, where the creation of the data sets is deeply detailed. Finally, it is explained the structure of the CNN, and the parameters employed for each layer.

2.1 The set up

The experiments were performed on an HP laptop, with an Intel Core i5 processor and 8Gb RAM. MATLAB was used to implement the deep learning algorithms, with the support of the Deep Learning Toolbox.

2.2 The dataset

Two image datasets were created to carry out experiments with the deep learning model. Each data set contain 1,193 binary images of handwritten digits, with a resolution of 28x28 pixels and a distribution of digits per class as shown in Table 1. In this paper, a 10-class handwritten digit classification problem is approached, where 750 images from each dataset are used (separately) as training examples (75 per class), and the rest of the images from both data sets are used (separately) as validation cases (443 images in total, where the number of images in the validation set are not equal per class).

Table 1. Digits per class in each dataset

Class	Examples	Class	Examples
0	107	5	108
1	169	6	101
2	122	7	100
3	149	8	123
4	100	9	114

<u>**Dataset 1**</u>

To create the dataset 1, we ask two different groups of undergraduates to write complete linear equations. Then, a 12 MP camera was used to take a series of RGB pictures of the linear equations written by students, as it is shown in Fig. 1. After image acquisition, the digits were segmented as can be observed in Fig 2. At the end, every segmented character constitutes a single image as depicted in Fig. 3. The stages of image segmentation and character extraction were done manually, using Microsoft Paint (a raster graphics editor). It is important to highlight that every character, whether is a digit, a sign or a letter, has a different resolution, which will be addressed and standardized following.

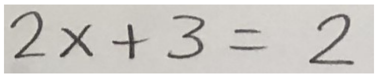

Fig. 1 Picture of a linear equation.

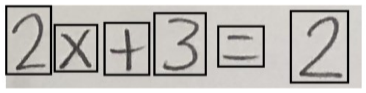

Fig. 2 Segmentation of the linear equation.

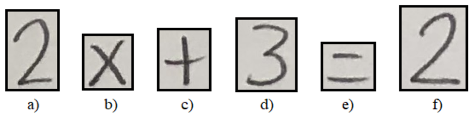

Fig. 3 Characters extracted from the linear equation.

Fig. 3 shows three types of characters extracted from the equation, where a), d) and f) are digits, b) is a letter and c) and e) are both math signs. In this paper, only digits are considered, since a digit classification problem is being addressed, which means neither letters nor math signs were contemplated. Each extracted digit was saved into a folder in their corresponding class, meaning there were 10 folders, 1 per class, and each one of them contained the digits distributed as shown in Table 1. Once all the digits were placed in the right class, the images were processed in MATLAB so the format of the original picture, see Fig. 4a, was changed from color to binary, as shown in Fig. 4b and then, their size could be standardized to a 28x28 resolution, see Fig. 4c. All 1,193 digits from the first dataset are binary images with a size of 28x28 pixels.

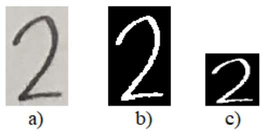

Fig. 4 Preprocessing of the original digits from the first dataset.

<u>Dataset 2</u>

To create the second dataset, a proposed method to preprocess images was applied to the dataset created in the previous section. As Fig. 5 illustrates, before it was applied a resize of 28x28 to the image all the edges were trimmed, see Fig. 5c, which means that every black row and black column were removed. This process was performed automatically on each image with MATLAB right before resizing it to a 28x28 resolution, see Fig. 5d.

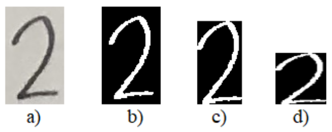

Fig. 5 Extraction of the characters of the linear equation.

3.3 The model

CNNs are networks with multiple layers, and they are used to solve image classification problems proved to perform well with unprocessed images. The CNN structure, regardless of its type or complexity, consists mainly of five layers [27]: an input layer, a convolutional layer, a pooling layer, a fully connected layer and a softmax layer.

An *input layer* is the input of the neural network. Before entering this layer, it must be decided how much the image or input should be preprocessed. Networks like LeNet-5, for example, work well on images with little preprocessing (as experiments show later).

The *convolutional layer* is the essential layer of the CNN. In this layer, the images are transformed in a set of representative features. The main objective is to reduce the images into something easier to process, without losing their important characteristics, which means creating a feature map. The element involved in carrying out the convolution operation in the convolutional layer is called kernel, filter or neuron, and this element, which is a square matrix smaller than the image itself, is in charge of taking square patches of pixels and passing them through the filter with a certain stride till the image is completely parsed, and significant patterns in the pixels are found. The convolution consists on taking the dot product of the filter with the patch of the image, where the value of the filters can assign importance to some aspects of the image, so they can be able to differentiate one image from other. These values will tend to be learnable values, called weights, and will be reinforced by some other learnable values, called biases, which are constant values. Equation (1) shows the convolution operation per filter.

$$\sum_i w_i\, x_i + b\,. \tag{1}$$

where *w* represents the weights, *b* the biases, *x* the inputs to the filters. Once the convolution operation is made, the decision whether or not the output of equation (1) is enough to "activate" or not the neuron is made by the activation function. There are some commonly used activation functions like those in the linear unit family, for example the ReLU function. The ReLU function has the characteristic of giving an output of zero for any negative input (or an input of zero), while giving the same output value to any positive input.

A down-sampling (data reduction) operation is performed in the *pooling layer*, where the size of the feature map is reduced. There are different ways of doing the down-sampling of the data; however, maxpooling is the usually used option.

The convolutional and down-sampling layers are followed by one or more *fully connected layers*, where the neurons connect to all the neurons in the preceding layer. The features of the image, learned by the previous layers, are combined to identify larger patterns, and the last fully connected layer combines the features to do the classification of the images. The number of outputs of the layer is equal to the number of classes in the target data (digit recognition is a 10-class recognition problem).

Lastly, in the *softmax layer* the softmax activation function normalizes the output of the fully connected layer. This layer consists of positive numbers that sum to one, which can then be used as classification probabilities by the classification layer.

The architecture used in this paper is a variation of the LeNet-5, as shown in Fig. 6, and it was decided to implement this type of CNN due to the nature of the character recognition problem (the LeNet-5 architecture was created to work specifically with a handwritten digit recognition problem). The setting of each layer is described below.

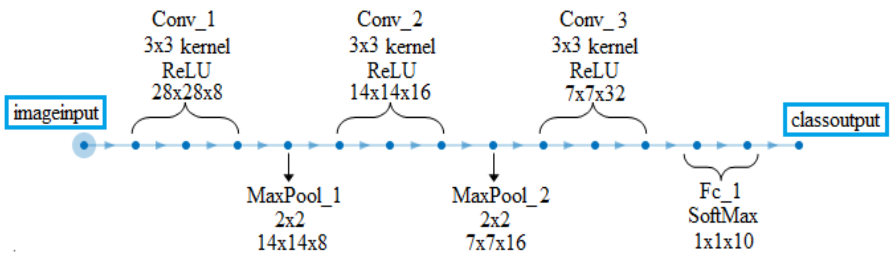

Fig. 6 Implemented CNN model.

- **INPUT.** The image size is 28x28x1.
- **Conv_1.** In the convolutional layer, it is applied a 3x3 kernel to the input. The layer has 8 filters, and it gives a 28x28x8 output, where 28x28 is the size of the convolved image and 8 is the number of features.
- **ReLU.** A Rectified Linear Unit is used as activation function in the CNN.
- **MaxPool_1.** In the max pooling layer, it is defined a 2x2 size for the rectangular pooling region where the layer gives a 14x14x8 output, where 14x14 is the size of the down-sampled data and 8 is the number of features.
- **Conv_2.** In the convolutional layer, a 3x3 kernel is applied. The number of filters of the layer is 16, which generates a 14x14x16 output, where 14x14 is the size of the convolved input of the layer and 16 is the number of features.
- **MaxPool_2.** In the max pooling layer, it is defined a size of 2x2 for the rectangular pooling region, and a 7x7x16 output of the layer is obtained, where 7x7 is the size of the down-sampled data and 16 is the number of features.
- **Conv_3.** In the convolutional layer, a 3x3 kernel is applied. The number of filters of the layer is 32, which generates a 7x7x32 output, where 7x7 is the size of the convolved input of the layer and 32 is the number of features.
- **Fc_1.** In the CNN model used, the fully connected layer works with 1,568 inputs, which are the result of the mathematical product of the flattened output given by the previous layer (7x7x32=1,568). The output size is 10, given that the number of classes in the target data is 10 (each per digit).
- **SoftMax.** The output of the SoftMax layer is a 10x1 vector with positive values that add 1.

- **classOutput.** After probabilities are obtained in the SoftMax layer, the class with the highest probability is activated and the image is classified.

In section 3, the experiments made with the CNN implemented and the classification of the two datasets formed are presented. Also, it is presented a discussion about the results and the response of the classifier.

3 Experiments and Results

In this section it is discussed the classification of the two different datasets used on the CNN model. The validation accuracy and training time were recorded at each run of the net, where the model ran on the same conditions for every test. There were 100 runs in total, 50 for each dataset, first with dataset 1 (digits shown in Fig. 4) and then with dataset 2 (digits trimmed, as shown in Fig. 5).

The accuracy and training time of each run (per dataset) were calculated, so it could be seen in general how often the classifier was correct predicting when a digit belongs to a class and when it does not belong, and how long it took the classifier to perform every test. Besides, it was important to see if an extra preprocessing to the image (the trimming on dataset 2) would cause a significant impact on the accuracy, the training time, or both.

Table 2 shows the validation accuracy and training time obtained with the dataset 1. As can be observed from Table 2, the lowest accuracy reached by the classifier was 94.58%, and the highest was of 97.74%, which means that out of all the predictions made by the classifier 97 out of 100 were correct. All training times were under 40 seconds, being 22 seconds the fastest. It is important noting that accuracy values and training time are not related at all, which means that highest accuracy values didn't necessarily give the fastest training times, or vice versa, which means that training times can't be used to predict how accurate the classifier will be. Fig. 7 shows visually all the accuracy and time values for dataset 1 (highlighting the lowest and highest validation accuracy values), and the average accuracy of the classifier (dash line). It can be seen on Fig. 7 that values of accuracy above average were obtained in more than half of the runs.

Table 2. Validation accuracy and training time for dataset 1

RUN	ACCURACY	TIME (s)	RUN	ACCURACY	TIME (s)
1	95.94	32	26	**97.74**	23
2	96.61	27	27	96.16	22
3	96.84	25	28	95.71	22
4	95.71	25	29	95.03	22
5	95.71	25	30	97.29	38
6	94.81	22	31	97.07	23
7	97.07	30	32	96.84	22
8	96.39	27	33	97.07	23
9	94.81	29	34	**97.74**	36
10	96.16	25	35	96.16	24
11	96.16	23	36	96.61	23
12	97.07	23	37	96.39	23
13	95.71	24	38	95.71	23
14	96.84	23	39	96.61	23
15	97.07	26	40	95.49	24
16	95.71	24	41	97.07	22
17	**94.58**	22	42	95.26	32
18	96.39	22	43	97.29	23
19	96.61	27	44	96.39	22
20	97.52	25	45	95.94	32
21	96.16	24	46	95.71	26
22	95.71	25	47	**94.58**	31
23	95.03	23	48	95.71	25
24	95.49	31	49	96.16	25
25	95.26	27	50	96.16	26

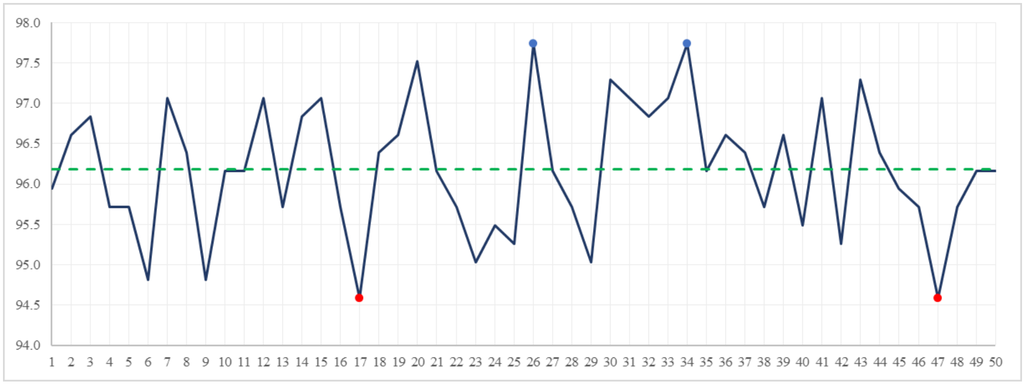

Fig. 7 Accuracy obtained with dataset 1.

Table 3 shows the validation accuracy and training time obtained with the dataset 2. The lowest accuracy reached by the classifier was 94.81%, and the highest was 98.19%, which means that out of all the predictions made by the classifier 98 out of 100 were correct. All training times were under 35 seconds, being 22 seconds the fastest. As seen in the experiments and results obtained with dataset 1, the accuracy values and training time are not related at all either. Fig. 8 shows visually all the accuracy and time values for dataset 2 (highlighting the lowest and highest validation accuracy values), and the average accuracy of the classifier. It can be seen on Fig. 8 that values of accuracy above average were obtained in more than half of the runs (dashline).

Table 3. Validation accuracy and training time for dataset 2

RUN	ACCURACY	TIME (s)	RUN	ACCURACY	TIME (s)
1	97.07	31	26	96.16	26
2	96.61	29	27	**98.19**	26
3	97.97	28	28	97.07	25
4	97.07	29	29	96.84	24
5	**98.19**	26	30	97.07	23
6	96.16	25	31	97.07	33
7	97.07	27	32	96.84	22
8	96.61	25	33	95.71	24
9	**98.19**	25	34	97.97	23
10	96.84	25	35	96.84	22
11	97.74	25	36	97.74	22
12	**98.19**	25	37	96.61	22
13	95.71	25	38	97.74	26
14	97.29	26	39	97.52	23
15	97.52	26	40	96.84	23
16	97.07	31	41	96.16	29
17	95.94	29	42	96.61	25
18	96.84	27	43	**94.81**	25
19	95.94	22	44	97.07	26
20	97.07	23	45	96.16	26
21	97.74	28	46	**98.19**	28
22	96.16	24	47	95.94	26
23	**94.81**	22	48	96.61	25
24	96.61	22	49	97.29	25
25	95.49	26	50	97.07	25

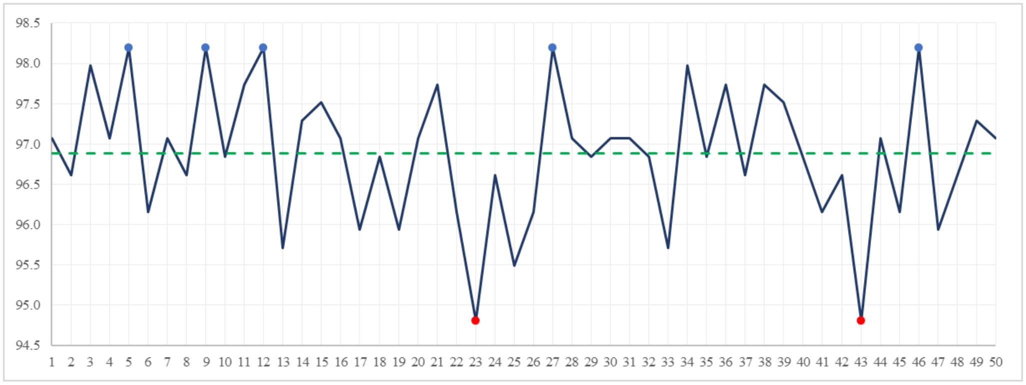

Fig. 8 Accuracy obtained with dataset 2.

A comparison between the accuracy and time values obtained by the classifier with both datasets is shown in Table 4. The minimum, maximum and average values of validation accuracy and training time are shown.

Table 4. Minimum, maximum and average values of accuracy and time

	DATASET 1		DATASET 1	
Value	**Accuracy**	**Time (s)**	**Accuracy**	**Time (s)**
min	94.58	22.00	94.81	22.00
max	97.74	38.00	98.19	33.00
average	96.19	25.42	96.88	25.50

Even though values obtained from both datasets are different, their minimum, maximum and average values of validation accuracy and training time are very similar, which could lead to believe that extra preprocessing in an image (or input in a CNN) is not necessary. However, the accuracy on the second dataset did improve (by 0.69 on the average calculated) and the training time was only slightly higher than the first dataset (by 0.08 on the average calculated), which could lead to believe (after all) that extra preprocessing in an image, in this case the edge trimming, may be not necessary but it can improve the performance of the classifier.

The experiments were made in order to analyze the behavior and results of a CNN, in the classification of two handwritten digit datasets, where it was applied to one of them an extra image preprocessing method: an edge trimming. According to the results of both the accuracy and training time obtained while classifying both datasets (with 1,193 images each), the proposed preprocessing method in this work (used on the images from dataset 2) gave higher accuracy values when compared to the values obtained from dataset 1 where images had not being extra preprocessed.

In summary, the achievements of this work were: 1) propose and create a new handwritten digit data set, 2) propose an image preprocessing method, applied to a set of images and create a new dataset with them, and 3) obtain higher accuracy values on the preprocessed set.

4 Conclusions

In this paper, a comparison of the performance of a CNN on a handwritten digit classification problem using two different datasets (created entirely in this work) was presented. It was used the same set up and general conditions for the testing of each function, and the experiments with the CNN were performed with each dataset separately.

In conclusion, according to the classifier results, it is possible to create a new handwritten digit dataset with little preprocessing and classify it with a CNN. It was viable to achieve accuracies above 94% with both datasets created (see Table 24), and it was possible to demonstrate that an additional preprocessing on an image (as an input on a CNN) can improve the performance of the classifier.

For future work, it is intended to develop and apply algorithms for (automatic) segmentation of the characters in the linear equations and it is considered to expand the experiments made on image classification beyond digit recognition, using every other character extracted from the linear equations collected, and beyond binary images. Also, a deeper analysis on the condition of unbalanced classes, and its impact on the performance of the classifier, is contemplated.

Acknowledgments. María Cristina Guevara thanks UACJ and CONACYT for support and scholarship to undertake her doctoral studies. Osslan Osiris, Vianey Cruz and Manuel Nandayapa thank UACJ for the economic support to carry out with success this investigation. Humberto Sossa thanks the Instituto Politécnico Nacional and CONACYT for the economic support under funds 20190007 and 65 (Fronteras de la Ciencia) to pursuit this research.

References

[1] M. Tanvir, A. Mahmoud, “Arabic handwriting recognition using structural and syntactic pattern attributes”, *Pattern Recognition*, vol. 46, no. 1, p. 141–54, 2013.

[2] H. Hamad, R. A. Zitar, “Development of an efficient neural-based segmentation technique for Arabic handwriting recognition”, *Pattern Recognition*, vol. 43, no. 8, p. 2773–2798, 2010.

[3] H. Lee, B. Verma, “Binary segmentation algorithm for English cursive handwriting recognition”, *Pattern Recognition*, vol. 45, no. 4, p. 1306–1317, 2012.

[4] A. Choudhury, A. Negi, S. Das, “Recognition of Handwritten Bangla Numerals using Adaptive Coefficient Matching Technique”, *Procedia Computer Science*, vol. 89, p. 764-770, 2016

[5] H. Sajedi, “Handwriting recognition of digits, signs, and numerical strings in Persian”, *Computers & Electrical Engineering*, vol. 49, p. 52-65, 2016.

[6] B. Q. Vuong, Y. He, S. C. Hui, “Towards a web-based progressive handwriting recognition environment for mathematical problem solving”, *Expert Systems with Applications*, vol. 37, no. 1, p.886–893, 2010.

[7] F. Álvaro, J. A. Sánchez, J. M. Benedí, “Recognition of printed mathematical expressions using two-dimensional stochastic context-free grammars”, *International Conference on Document Analysis and Recognition*, p. 1225-1229, 2011.

[8] Y. Chajri, B. Bouikhalene, “Handwritten mathematical expressions recognition”, *International Journal of Signal Processing, Image Processing and Pattern Recognition*, vol. 9, no. 5, p. 69-76, 2016.

[9] M. Edwin, W. Putra, “Structural Off-line Handwriting Character Recognition Using Approximate Subgraph Matching and Levenshtein Distance”, *Procedia Computer Science Elsevier*, vol 59 , p. 340-349, 2015.

[10] J. Dasgupta, K. Bhattacharya, B. Chanda, “A holistic approach for off-line handwritten cursive word recognition using directional feature based on arnold transform”, *Pattern Recognition Letters*, vol. 79, p. 73-79, 2016.

[11] T. Wakahara and Y. Yamashita, “K-NN classification of handwritten characters via accelerated GAT correlation”, *Pattern Recognition*, vol. 47, Issue 3, pp. 994-1001, 2014.

[12] M. Kumar, M.Jindal, R. Sharma, “KNN based offline handwritten Gurmukhi character recognition”, *In Image Information Processing (IcIIP)*, p. 1-4, 2011.

[13] C. Malon, S. Uchida, and M. Suzuki, “Mathematical symbol recognition with support vector machines”, *Pattern Recognition Letters*, vol. 29, no. 9, p. 1326–1332, 2008.

[14] M. Elleuch, M. Kherallah, “An Improved Arabic Handwritten Recognition System using Deep Support Vector Machines”, *International Journal of Multimedia Data Engineering and Management*, vol. 7, no. 2, p. 1-14, 2016.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks”, *in Neural Information Processing Systems*, p. 1097–1105, 2012.

[16] S. R. Kulkarni, B. Rajendran, “Spiking neural networks for handwritten digit recognition-supervised learning and network optimization”, *Neural Networks*, p. 103, 118–127, 2018.

[17] X. Niu and C. Y. Suen, “A novel hybrid CNN-SVM classifier for recognizing handwritten digits”, *Pattern Recognition*, vol. 45, no. 4, p. 1318-1325, 2012.

[18] M. Elleuch, N. Tagougui, and M. Kherallah, “A novel architecture of CNN based on SVM classifier for recognizing Arabic handwritten script”, *International Journal of Intelligent Systems Technologies and Applications*, vol. 15, no. 4, p. 323–340, 2019. Preprint, arXiv:1712.03541v2.

[19] Y. LeCun, B. Boser, J. S. Denker and D. Henderson, “Backpropagation applied to handwritten zip code recognition”, *Neural Computation,* vol. 1, no. 4, p. 541–551, 1989.

[20] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard and L. D. Jackel, “Handwritten digit recognition with a back-propagation network”, *Advances in Neural Information Processing Systems,* vol. 2, pp. 396-404, 1990.

[21] A. Krizhevsky, I. Sutskever and G. E. Hinton, “Imagenet classification with deep convolutional neural networks”, *NIPS,* pp. 1106-1114,

2012.

[22] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, “Gradient-based learning applied to document recognition”, *Proceedings of the IEEE,* vol. 86, no. 11, pp. 2278-2324, 1998.

[23] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov and D. Erhan, “Going deeper with convolutions”, *CVPR,* pp. 1-9, 2015.

[24] Y. LeCun, C. Cortes and C. JC Bur, “MNIST handwritten digit database”, AT&T Labs, 2010. Available: http://yann. lecun. com/exdb/mnist 2.

[25] Y. LeCun, C. Cortes, C. Burges, “The mnist database”, 1998, http://yann.lecun.com/exdb/mnist/

[26] A. Krizhevsky, V. Nair, G. Hinton, “The CIFAR-10 dataset”, 2009, https://www.cs.toronto.edu/~kriz/cifar.html/

[27] L. Guifang and S. Wei, “Research on convolutional neural network based on improved Relu piecewise activation function”, *Procedia Computer Science,* vol. 131, pp. 977-984, 2018.