Novel Preprocessors for Convolution Neural Networks

ZIAD DOUGHAN, ROLA KASSEM, AHMAD M. EL-HAJJ, AND ALI M. HAIDAR

Department of Electrical and Computer Engineering, Beirut Arab University, Debbieh, Beirut 1107 2809, Lebanon

Corresponding author: Ziad Doughan (z.doughan@bau.edu.lb)

ABSTRACT Fooling neural networks is a main concern in the process of Artificial Intelligence optimization. Character perturbation make part of a text unnoticeable for some systems, even for human observers. This research focuses the probe on a novel input preprocessing technique, which applies with the Convolutional Neural Networks for character recognition applications. Using Hand-written, Font, and Arabic Alphabet datasets, we show the CNN competitive results in classification. Then, we apply random deformation by mean of reshaping and relocation on the three testing datasets. As a reaction to change, the CNN demonstrates a confused behavior. Therefore, we design a Weightless Neural Network preprocessor that optimizes the shapes of deformed characters. Finally, we compare the impact of corrected input in improving the CNN performance. To measure the success of our preprocessor, we have applied it on well-known CNNs to compare the results between defected and rectified test datasets. Using Keras, we reconstructed LeNet-5, AlexNet and ZFNet for testing. Then, we conduct the test on the three databases to show how the recognition capability of CNNs would decline, then improve after preprocessing.

INDEX TERMS Artificial neural networks, logic-based intelligence, pattern recognition, preprocessor, weightless neural networks.

I. INTRODUCTION

The Artificial Neural Network (ANN) progress has transformed both Machine Learning (ML) and Artificial Intelligence (AI) to a new perspective [1]. Many applications proved the efficiency of ANNs in extracting information [2]. Even with this impressive capability, ANNs remain quite under exploration to overcome the difficulty of typifying network, and to understand how its architecture evolved from data [3]. As ANN systems are making an apprehensive impact on the real world, much stress is rising upon the operating methods of those complex models [4]. Transparency in such applications is pressing for more exact and consistent decision results, typically in medicine, policies and economy [5].

LeCun breakthrough led the Convolution Neural Networks (CNN) to a state-of-the-art technology for many challenges. Its idea was to create a deep hierarchical structure capable of filtering weights across the network. The multi-layer pipeline learns to detect several patterns from the processing operation of the training dataset. The captions acquired by the early layers are associated with scores in the output layers [6].

The associate editor coordinating the review of this manuscript and approving it for publication was Wei Wang.

Maybe CNNs present a state-of-the-art performance in image processing but they are far from a “True” object detection approach. CNNs are highly sensitive to any single parameter disturbance. This noise can be in many forms like shapes, colors, intensity, etc. To aide CNNs in their understanding to these distortions, training included shape shifting, color changes, reflection, and shadows [7]. However, more works demonstrate that text classifiers collapse with the same encounters used for image attacks [8]. Therefore, attackers tried to target the classification mechanism of CNNs by many ways. And lately, the robustness of CNNs test its capabilities by processing adversarial samples, which can cause wrong classifications [9].

In contrast, humans can observe and classify things correctly even without noticing several glitches in the observed object [8]. As cognitive response remains the main scope of interest for many researchers, one should expect harder challenges. Explorations of pictures using ANN systems face multiple problems, as if for example shape-shifting scenarios can render the systems stranded. Therefore, a wide crowd expect that a Boolean system is a more promising manner to provide influential ANN prototype [10].

On the other hand, one of the distinguished ANN prototypes is the Weightless Neural Networks (WNN).
This logical neuron model uses a simpler operation approach, which consist of a look-up table mechanism. Designers can realize WNN using RAMs to save the values in memory cells instead of calculating them. The implicit approach of memory addressing allows WNN to model innovative ML systems, due to its aspect of architectural expansion [10].

Inspired by the human’s eye focusing mechanism and relying on the WNN binary operation, the proposed Preprocessor, of this research, is suitable to redraw the input in a more exact representation. Mainly, we focus on two applications, a Translation to position the character and a Scaling to fit it exactly to the CNN input size. The character’s reconstruction, based on a neural network that preserves key features, is important to provide a high-definition input for the CNN. Rich information existing in the textures processed persist well in the compressed figures.

The main work of this research is to fool well-known CNN models by applying data augmentation on some traditional handwritten datasets. When trying to predict the characters, the CNN models are expected to fail in this task. One of the main approaches to solve this problem is to re-train the CNN model on the augmented dataset. But instead of using this approach, our target is to equip those CNN models with a multistage preprocessor that will regenerate the character in a familiar form to the CNN. Thus, we concentrate our preprocessor design to redraw the character feature in their expected location on the input canvas. Afterward, we test the enhanced CNN against the same manipulated dataset using in the fooling process and show how the prediction results are improved.

The most innovative point in this research is making data argumentation processing, executed in the WNN preprocessor, as a separate module. Thus, the idea is the decoupling of the data argumentation processing from the full sequence of CNN processing. This separated model designed using ram based WNNs emphasize its importance of being deployment friendly when it comes to embedded systems and robotic applications. Thus, a certain robot containing a CNN model enhanced with such preprocessor may overcome may prediction obstacles without any retraining activities.

The paper continues as follows: In section II we explore previous works. Section III stages the experimental road map methods and network architectures designed in this research. Section IV presents the results of the novel preprocessor in optimizing the input of a CNN. In section V, we discuss the impact of fooling CNN, which transmits a down fall of test’s results in contrast with enhanced figures. We conclude our work in Section VI with future scope.

II. PREVIOUS WORKS
Object classification is a popular application in ANN. Usually, ANNs excel in fitting large dataset to classify a single object, where the network directly forecast class scores [11]. First, the network operates to emphasize specific odd patterns. It is often hard to lead the CNN training to a clear regression on which pattern to use or ignore [4]. Therefore, the resulting CNN does not cover in its design all odd appearances of an object [11]. Research showed that small changes in object projection could cause a perturbing result. For example, a rotation of an object can hit on a network and cause total confusion. Likewise, feature translations or resizing can lead to the same effect [7].

Second, CNNs are targets to fool assumption. One can tolerate it for simple gesture application, but cannot accept it for critical assessments [11]. In recent past, long short-term memory (LSTM) and recurrent neural networks (RNN) seemed unfeasible for self-evolving systems. However, today several heuristic systems optimize their results using those nets where the long-lasting effect of the self-loop can rectify onetime appearing input glitches. Thus, RNNs’ works support the strategy of using weight matrices to promote optimizations [12].

Third, to improve these vulnerabilities, the training process may include multiple positions of the object. This approach proved efficient on MNIST datasets, but still ineffective for others like CIFAR10 and ImageNet [13]. Therefore, more advanced methods focus on “random input transformation” techniques to insure robustness. However, it requires a huge training process overhead to achieve notable enhancements. In some cases, results may not be pleasant even after a long and exhaustive modeling exercise [13].

Finally, artificial perturbation samples can easily mislead a CNN model. Many attack techniques revealed the sensitivity of these networks on certain features in a dataset [8]. On the other side, a theoretical outlook believes that multiple class models are more specific, while single class are more general. Thus, it is more complex but more powerful to use multiple single label classifiers in parallel for multi-label classification models [14]. However, this can be realized if categories are independent form each other. Conversely, in nature this situation is uncommon. Therefore, other strategies, like kernel methods and decision trees, proved more successful in completing ANN systems [15].

III. METHODS AND DESIGN
It is more complicated to design a CNN that carry out the detection of every possible misleading feature. Often, when training CNNs, it is easy to perform 80% to 90% accuracy, but it may be almost impossible to reach the rest correctly. Furthermore, there is today a great concern to preserve the achieved performance when a neural network is being fooled by data manipulation. Thus, we choose in our experiment well-known CNNs, which will fail when predicting the perturbed dataset prepared to fool them. Also, we have decided on purpose to choose well-known network exhaustively studied with a well-known dataset (MNIST handwritten dataset) to prove their vulnerability. Furthermore, we choose on purpose to enhance the CNNs with a novel preprocessor, which works with many CNN models and improves the predictions of the CNNs without any re-training on augmented dataset. Although, re-training CNNs with data augmentation may solve the issue, but it is meaningless to re-train the model each
time some attackers find a way to manipulate data. Therefore, to achieve best recognition abilities, it is more favorable to perform data preprocessing before feeding it to the CNN. Thus, the preprocessor aims to optimize and unite distinctive features of each character to generate a pattern set.

Figure 1 shows the road map of the testing implementation. First, we train LeNet-5, AlexNet, and ZFNet CNNs on the regular training datasets. The next three steps work to generate three results for a regular, deformed and optimized datasets. A random reshaping agent distorts the testing datasets and the WNN preprocessor rectifies the deformed characters. The convolution networks will eventually deliver better results on an optimized dataset.

This preprocessor provides a new variant approach in CNN empowerment, a training free optimization method. In which, the preprocessor makes the inputs more suitable to the CNN model instead of a repetitive remodeling mechanism[16]. Unlike other optimization methods, this privilege allows robots and machines to have embedded chips that alternate their readings. We successfully design multiple WNNs that tune the characters in different variations. It is like providing the machine with a mimic of an eye focusing mechanism that clips the character in any location or size as needed. The essential is to organize the input optimization sequence in order to serve best the required application. Therefore, in our test we aim to fully cover the input canvas of the CNN to provide best character classification.

Figure 3 shows a diagram of the preprocessor used in the CNN input optimization. The preprocessor consists of four consecutive stages that optimizes the input form. While the translator resolves the arbitrary location that a character appears at. The input is first processed by two row and column Translators of size 28 x 28. Followed by two row and column Scalers of the same dimension. The scaler redistributes the character size over the whole canvas. Finally, the Averager’s layer smoothen the presentation of the character and the ReLU work on leveling and intensifying the pixels.

**A. THE TRANSLATOR**

The form optimization passes two networks, the Translator and the Scaler. The shifting and scaling mechanism presumes a transitory stage to transform the character in a better fit to the correlation of the symbolic representation. Thinking of it from another perspective, the extracted group of pixels from a specific location in CNNs is summed up together. Then, the system moves to the next set using a sliding filter. It is operational effective implying an invariant alteration when retrieving pixel patches within the same space[4].

Therefore, the ANN operates with MNIST characters sequentially, and significant patterns classify the output.
Considering those two stages, the pixels take a fine place between the four corners of the working canvas. The random distribution of the pixels optimizes to a better mapping over the generated output matrix. Therefore, for the same digit, multiple models’ styles add to each other, as expected, to generate the ideal representation for each category.

Figure 4 shows the pipeline and the diagram of the Translator shifting the character horizontally and vertically. The target of the Translator is to shift the group of pixels in x-direction then in y-directions sequentially. The dimension of the image in the pipeline is 32 x 32 pixels. The WNN moves the symbol until the left most pixels reach the column control neuron. By default, the halfway output and the network output dimensions are also equal to 32 x 32 pixels. The five-layer architecture operates using simplistic hardware. The accumulation layers consist of ram grid to hold and bypass the values of each stage.

![FIGURE 4. The translator neural network diagram.](image)

In more detail, the translation layers consist of a grid of RAMs that generates a binary output based on an address selected. As shown in the blue dotted zone of Figure 4, the neural RAM has four inputs, which implies sixteen possible output selections. The key difference between ANN and WNN is the memory grid size instead of the weights complexity [17]. However, in this model the typical functionality of the neural nodes allows to use the same neuron in series instead of parallel processing. However, the system’s speed will decline to shrink the architecture and provide a compact network.

In mathematics, the translation function in image processing usually considers the horizontal and vertical position of the object visualized [18]. Therefore, in our methods we provide a WNN grid that provides the same approach in its inner layer. Most of the weightless neurons’ output targets the previously adjacent neuron in order to produce a recurrent network layer, which seeks to shift the image one-step at a time. The proposed structure, which moves the image to the top left most location of the layer, provides an aligned repositioning mechanism for any input received. The purpose of this approach is to present a compact hardware that operates on the pixels level and contribute to any possible image processing system.

The activation control unit start the recurrent mapping application, which stops when the image reaches the marginal nodes, and the output is ready for retrieval. Each neuron in the translator is a RAM that loads specific values in each address. Consequently, the RAM generates those values based on an address formed by a combination value of the input, the activation, the control, and the recurrent adjacent output. Figure 5 shows the interconnection between the WNN RAM units of a column shifting Translator. The address lines of the Translator WNN RAM are connected to \( i_c \) (input control), \( c \) (network control), \( i_{c+1,y} \) (input values), and \( o_{x+1,y} \) (recurrent output of adjacent neuron). The output \( o_{x,y} \) is the RAM retrieved value at the selected address. The input control WNN and the column control WNN units are unique. While, for each pixel in the 32 x 32 image there exist an input WNN and a column Translator WNN to operate the desired translation function. Similarly, a row shifting Translator have the same connectivity but in a vertical association.

![FIGURE 5. A translator network using WNN RAMs.](image)

The training phase, for each WNN, corresponds of loading the set of outputs \( S_o = O_0, O_1, \ldots, O_n \) to its associated address locations in the RAM representing the WNN. When all the Output sets are loaded to the RAMs in the network layers, the system is ready for operation [14]. Therefore, we have employed a set of rules to define the equation of a Translator:

- To load the pixels’ values from the input layer: if the input control is set to “High” (\( i_c = 1 \)). Then, the output \( o_1 \) is equal to the input \( i \).
  \[
  o_1 = i_c \land i \tag{1}
  \]

- To shift the pixels in the translation layer: if the input control is set to “Low” (\( i_c = 0 \)) and the shift control is set to “High” (\( c = 1 \)).
  \[
  o_2 = ¬i_c \land c \land o_{x+1,y}[t-1] \tag{2}
  \]

Unifying the two conditions in one function: if one or both equations (1) or (2) are met, then the output \( o \) is generated corresponding to \( o_1 \) or \( o_2 \). Accordingly, the general equation (3) of the Translator is as following:

\[
 o_t \left[ t \right] = (i_c \land i) \lor (¬i_c \land c \land o_{x+1,y}[t-1]) \tag{3}
\]

**B. THE SCALAR**

The second stage of the optimization is the Scaler, which is more complicated than the previous network. The scope of the Scaler is to pose the character in the input on the canvas.
to eliminate irrelevant placement variations. In deep learning models, to achieve great success in visual applications, a robust learning capability is required to fulfill tasks like classification and detection [19]. Therefore, redistributing the characters on the full canvas improves the spatial occurrence of a specific feature in a character image. Figure 6 shows a diagram of the Scaler identified by two main layers, a column and a row resizing WNN. The Scaler operates on two directions horizontally and vertically using a bilinear interpolation mechanism. To implement this method, the Scaler consists of five sub networks. While, the Scaler is not based on a specific factor, the network remains stretching the image until it fills the whole canvas. The WNNs layers’ dimension is 32 x 32. Thus, the scaling process does not preserve the aspect ratio of the character. The network automatically redraws the character rows and columns to stretch it without maintaining its aspect ratio.

The Scaling mathematical function implementation necessitates a lot of effort to model such application in digital logic designs. The potentials developed in WNNs allows an easier approach towards modeling such system [12]. However, it still needs a switching control that relatively explore the right location, in which the stretching pixels take place. We find by experience that the process attends to insert row at a midway distance each time. At initialization, the process inserts a row at the middle of the canvas, then one at each quarter, then one at each sixteenth, and so on until between any two original rows there is a new row inserted. At each insertion, the system detects if the image has reached the edge of the canvas to stop the iterations. If the image has doubled in size but did not reach the maximum edge, then network starts over again.

The key insight in WNN model is that instead of comparing long data set features via regressions, one can find an operational measure to set models structures [2]. As existing evidence of this robustness in WNNs, Figure 7 shows the interconnection between the Scaler neurons. While the input control WNN and the column control WNN units are unique in the system, the Column Integration Control is the couple ($O_{cb}, O_{ci}$), for each column in the Scaler layer. For each pixel in the 32 x 32 image, there is an input WNN and a column Scaler WNN to do the stretching. Likewise, a vertical architecture of the same connectivity represents a row Scaler model. The address of the Scaler WNN RAM is connected to $i_c$ (input control), $c$ (network control), $o_{x,y}$ (input values), $o_{x-1,y}$ (recurrent output of adjacent neuron), $o_{x,y}$ (recurrent output of the same neuron), $o_x$ (integration control), and $o_{cb}$ (shifting control). The output $o_{x,y}$ is retrieved memory value from the RAM at a specific 7-digits address. The repetitive nature of this image processing operation allows building neural layers of similar structures. The inner core of the network shows that its architecture can expand or shrink based on the resources available for the design. The final product is easily implementable in multiple types of hardware.

\begin{align}
\text{FIGURE 6. The scaler network diagram.}
\end{align}

The neural control grid of the Scaler is about the object reaching the borders of the canvas. There is no need to a coefficient to set the new scale of the character, however this automatic tool does not consider any aspect ratio preservation. This matter is customized intentionally to reallocate the character features spatially. For example, the number two is defined by a half circle on the top traversing from the left to the right. The operation of the Scaler is to insure that in the CNN’s learning phase or operational phase this feature falls in the same place for comparison. Then, the decision, whether the CNN points to the right character or not, improves using this caption tool.

Next, we discuss the Scaler function used to generate the stretched image samples. As seen in the Scaler structure, the WNN RAM take seven address lines to produce a specific output. We argue logically the set of rules required to control such function consecutively as input loading, row integration, and halting. This sequencing paradigm enables a swift “shift-insert” mechanism to do the trick:

- To load the pixels’ values from the input layer: if the input control is set to “High” ($i_c = 1$). Then, the output $o_1$ is equal to the input $i$.
  \begin{equation}
  o_1 = i_c \land i
  \end{equation}

- To halt the stretching and maintaining the same pixels in the WNN: if the input control is set to “Low” ($i_c = 0$) and the stretching control is set to “High” ($c = 1$). Then, the output $o_2$ remain with the same value in the neuron at time $t - 1$.
  \begin{equation}
  o_2 = \neg i_c \land c \land o_{x,y}[t-1]
  \end{equation}
Thus, the general equation (9) of the scaler is as following:

\[ o_3 = \neg i_c \land \neg c \land o_{cb} \land o_{x-1,y}[t-1] \]  

(6)

- To insert a new pixel in between of two existing pixels: if the input control is set to “Low” \( (i_c = 0) \) and the stretching control is set to “Low” \( (c = 0) \), and if the column shifting control is set to “High” \( (o_{cb} = 1) \). Then, the output \( o_4 \) is equal to either one of the two pixels in process \( o_{x,y}[t-1] \) or \( o_{x-1,y}[t-1] \):

\[ o_4 = (\neg i_c \land \neg c \land o_{ci} \land o_{x,y}[t-1]) + (\neg i_c \land \neg c \land o_{ci} \land o_{x-1,y}[t-1]) \]  

(7)

- For the pixels that falls before the insertion point and do not change values while the network is operational: if the input control is set to “Low” \( (i_c = 0) \) and if both the column shifting and integration controls are set to “Low” \( (o_{cb} = 0 \) and \( o_{ci} = 0) \). Then, the output “o5” remain with the same value in the neuron at time “t-1”.

\[ o_5 = \neg i_c \land \neg o_{cb} \land \neg o_{ci} \land o_{x,y}[t-1] \]  

(8)

Unifying all the previous conditions in one function: if one or many of the equations from (4) to (8) are met, then the output \( o \) is generated corresponding to \( o1, o2, o3, o4, o5 \). Thus, the general equation (9) of the scaler is as following:

\[ o[t] = (i_c \land i) \lor (\neg i_c \land c \land o_{x,y}[t-1]) \lor (\neg i_c \land \neg c \land o_{cb} \land o_{x-1,y}[t-1]) \lor (\neg i_c \land \neg c \land o_{ci} \land o_{x,y}[t-1]) \lor (\neg i_c \land \neg o_{cb} \land \neg o_{ci} \land o_{x-1,y}[t-1]) \lor (\neg i_c \land \neg o_{cb} \land \neg o_{ci} \land o_{x,y}[t-1]) \lor (\neg i_c \land \neg o_{cb} \land \neg o_{ci} \land o_{x-1,y}[t-1]) \lor (\neg i_c \land \neg o_{cb} \land \neg o_{ci} \land o_{x,y}[t-1]) \]  

(9)

To finalize the design of the scaler and command the rhythm of the insertion mechanism, we have used a column integration control unit. This controller is a set of insertion sequences each at a point to evenly distribute the stretching columns in between original ones. This crown jewel represents the heart of the scaler, which replicates a Bilinear Interpolation function.

Figure 8 shows a diagram of the scaler integration control unit. This network consists of three layers, the column sequence WNN layer, the column integration layer, and the column shifting layer. Whereas the column sequence WNN specifically cast a well define series of insertions, the integration and the shifting networks decode this series and stretch the image until it fits to the canvas. This controller connects to all the rows of the scaler layer. The cells in green represent a scaler row example composed of 32 WNN RAM. The network automatically controls the scaler rows by two means. First, the column shifting control units activate by a signal from either a column integration unit or a previous shifting control unit. Second, a column integration unit receives its activation from the column sequence WNN to perform the integration. Each one of the control units is connected to the scaler rows’ units in the same manner all along each column. For example, the integration unit “\( Q_{ci}[1] \)” is connected to each scaler unit in the column at position 1. From the other side, the shifting unit “\( Q_{cb}[2−3] \)” is connected to all the neurons in both columns 2 and 3.

\[ \text{FIGURE 8. An integration control unit of a column scaler.} \]

\[ \text{FIGURE 9. The averager network diagram.} \]

C. THE AVERAGER

The concept of similarity detection between the sample dataset improves by the interception of the same features within the same space, using this translating and resizing optimization. Considering that the neural network canvas is divided to smaller regions, each region must be smoothened to insure a wider overlapping space between features. For this purpose, the averager operates to level the pixels of the character in the learning dataset. As shown in Figure 9, this network consists of multiple weightless neural layers. The blue zone illustrates how an averager neuron captures several pixels in a neighborhood, then return an output pixel reproduced from the median of the input. The tiny parts of the image are discarded, leaving a neat output at the end. The first layer receives the optimized character from the previous stages. The core layer of the averager in its turn work to generalize the features addressed by the CNN. On the output layer, the character is fuzzier and bolder.

Its mathematical function has a long history with ANN models especially for image classification and is present wildly in CNN applications generally in pooling layer [20].
Like other networks, we apply the WNN technology to design the Averager. This realizes two goals: a smoothing effect and pixel-level rectification. Thus, it performs an essential task required in convolution nets.

The Averager layer consists of multiple WNN units each connected to a patch of 3 x 3 input neurons. Thus, each Averager unit connects to nine input units and to one input control unit, making the address length of the RAM 10-bits. Therefore, we divide the Averager structure into two stages, a row averaging layer, and a column averaging layer. The WNNs of the first layer have a 4-bits address connected to three input neurons and generate a 2-bits output. The WNNs of the second layer have an input address of 6-bits connected to three WNNs from the first layer and generate the final averaged output.

Figure 10 shows the connections of the Averager system. The two layers composition network allows reducing the address size of the Averager WNNs. As displayed, the Row Averager WNNs connect to a set of inputs within the same row. Consecutively, a Column Averager sums up all the results of the Row WNNs. Likewise, an input control \( i_c \) also controls the first layer. Finally, the output \( o_{x,y} \) is the result of averaging a patch of inputs in the network. Each set of inputs with a size of 3 x 3 designate an input patch for the Averager units. A generic design of an Averager unit consists of having 10-bits address RAM. Thus, having \( 2^{10} = 1024 \) memory cells per WNN. Therefore, we propose a two-layered structure, which requires three Row Averager WNNs with 4-bits address and one Column Averager WNN with 6-bits address. In a simple mathematical calculation, this leads to \( 3 \times 2^4 = 48 \) plus \( 2^6 = 64 \), which is equal to 112 memory cells per four WNNs, reducing the memory size by 912 cells.

![An averager network using WNN RAMs.](image)

The cascading structure does not only reduce the RAM size but also transforms the averaging function to a bit counter in this neural network. In sequence, the input is loaded to the Averager layer to be processed when the control input \( i_c = 1 \). There are no recurrent operations in this network. Therefore, the first layer WNNs count the existing “1s” in each row. Accordingly, the column neuron sums up the 1s occurrences to either generate 1 for “5 occurrences and above” or 0 otherwise. Each input patch of 3 x 3 neurons leads to an output, except the bordering output that remains at 0 values. With an overlapping connectivity, the Averager produces a 32 x 32 output at its final stage.

To improve the process, we define the following rules required to control the filtering structure that leads to a smoother character representation. The “counter-adder” composition allows flattening the symbol to a wider mapping space making its features more general than specific:

- To load the pixels’ values from the input layer to the row Averager WNN and generating its lower bit: if the input control is set to “High” \( (i_c = 1) \). Then, the output lower bit \( AL_y \) is equal 1 if the 1s count is odd or 0 if the count is even. Therefore, \( AL_y \) is as following:
  \[
  AL_y = i_c \land i_{x-1,y} \land \neg i_{x,y} \land \neg i_{x+1,y}, \\
  \lor (i_c \land \neg i_{x-1,y} \land i_{x,y} \land \neg i_{x+1,y}), \\
  \lor (i_c \land \neg i_{x-1,y} \land \neg i_{x,y} \land i_{x+1,y}), \\
  \lor (i_c \land i_{x-1,y} \land i_{x,y} \land i_{x+1,y})
  \]
  \( (10) \)

- To load the pixels’ values from the input layer to the row Averager WNN and generating its higher bit: if the input control is set to “High” \( (i_c = 1) \). Then, the output higher bit \( AH_y \) is equal 1 if the 1s count is greater or equal to two or 0 if other wise. Therefore, \( AH_y \) is as following:
  \[
  AH_y = i_c \land i_{x-1,y} \land i_{x,y}, \\
  \lor (i_c \land \neg i_{x-1,y} \land i_{x,y} \land \neg i_{x+1,y}), \\
  \lor (i_c \land \neg i_{x-1,y} \land \neg i_{x,y} \land i_{x+1,y}), \\
  \lor (i_c \land i_{x-1,y} \land i_{x,y} \land i_{x+1,y})
  \]
  \( (11) \)

- To load the values from the counter WNNs to the adder WNN in the Averager and generating its output \( o_{x,y} \): if the sum of the numbers from the counters are greater or equal to five then \( o_{x,y} = 1 \), otherwise \( o_{x,y} = 0 \). Therefore:
  \[
  o_1 = (AL_{y-1} \land AH_{y-1} \land AH_y) \\
  \lor (AL_{y} \land AH_{y} \land AH_{y+1}) \\
  \lor (AL_{y} \land AH_{y} \land AH_{y-1}) \\
  \lor (AL_{y+1} \land AH_{y+1} \land AH_{y})
  \]

If one of the rows has three 1s and another has two or more, then:

\[
  o_1 = (AL_{y-1} \land AH_{y-1} \land AH_y) \\
  \lor (AL_{y} \land AH_{y} \land AH_{y+1}) \\
  \lor (AL_{y+1} \land AH_{y+1} \land AH_{y})
  \]
If one of the rows has three 1s and the other two has one each, then:

\[ o_2 = (AL_{y-1} \land AH_{y-1} \land AL_y \land AL_{y+1}) \]
\[ \lor (AL_y \land AH_y \land AL_{y-1} \land AL_{y+1}) \]
\[ \lor (AL_{y+1} \land AH_{y+1} \land AL_{y-1} \land AL_y) \]

If two of the rows has two 1s each and the other one has one, then:

\[ o_3 = (AL_{y-1} \land AH_y \land AH_{y+1}) \]
\[ \lor (AL_y \land AH_y \land AH_{y-1}) \]
\[ \lor (AL_y \land AH_y \land AH_{y+1} + 1) \]
\[ \lor (AL_{y+1} \land AH_{y+1} \land AH_{y-1}) \]
\[ \lor (AL_{y+1} \land AH_{y+1} \land AL_y \land AH_y) \]
\[ \lor (AL_{y-1} \land AH_{y-1} \land AL_y \land AL_{y+1}) \]
\[ \lor (AL_y \land AH_y \land AL_{y-1} \land AL_{y+1}) \]
\[ \lor (AL_{y+1} \land AH_{y+1} \land AH_{y-1} \land AL_y) \]
\[ \lor (AL_{y-1} \land AH_{y-1} \land AH_{y+1}) \]
\[ \lor (AL_y \land AH_y \land AH_{y+1}) \]
\[ \lor (AL_{y+1} \land AH_{y+1} \land AH_y) \]
\[ \lor (AH_{y-1} \land AH_y \land AH_{y+1}) \]

(12)

If the three rows has two or more 1s, then:

\[ o_4 = (AH_{y-1} \land AH_y \land AH_{y+1}) \]

The above functions cover all the possible cases that lead to a “5+ occurrences” of 1s, to generalize the output if we sum up all the above we get the following equation:

\[ o_{x,y} = (AL_{y-1} \land AH_{y-1} \land AH_y) \]
\[ \lor (AL_y \land AH_y \land AH_{y-1}) \]
\[ \lor (AL_y \land AH_y \land AH_{y+1} + 1) \]
\[ \lor (AL_{y+1} \land AH_{y+1} \land AH_{y-1}) \]
\[ \lor (AL_{y+1} \land AH_{y+1} \land AH_y) \]
\[ \lor (AL_{y-1} \land AH_{y-1} \land AL_y \land AL_{y+1}) \]
\[ \lor (AL_y \land AH_y \land AL_{y-1} \land AL_{y+1}) \]
\[ \lor (AL_{y+1} \land AH_{y+1} \land AL_{y-1} \land AL_y) \]
\[ \lor (AL_{y-1} \land AH_{y-1} \land AH_{y+1}) \]
\[ \lor (AL_y \land AH_y \land AH_{y+1}) \]
\[ \lor (AL_{y+1} \land AH_{y+1} \land AH_y) \]
\[ \lor (AH_{y-1} \land AH_y \land AH_{y+1}) \]

(12)

Finally, the three sub-networks of the Preprocessor operate so that the confidence after the training sessions of a CNN correspond to the desired levels. Inevitably, this Preprocessor will add some time consumption on the training dataset enhancement. However, the distinguishing features of the patterns are likely to be obtained sharply than the rest.

### IV. RESULTS

One of the best choices was to involve well-known CNNs in the analysis phase. The results’ comparison of the predictions of the LeNet-5, AlexNet, and ZFNet against the regular, reshaped, and optimized data will prove the core objective of our research. As recognized, a Convolutional network consists of multiple convolution phases including convolution layers followed directly by pooling layers. Accordingly, the network passes the convolutions result to a stack of flattened layers followed by a Softmax classification [18]. For each of the character’s samples, the CNN processes three variations.

Table 1 shows the Python setup used in implementing LeNet-5 CNN. The section on top presents the parameters used in initializing each layer. In addition, it exhibits the model summary of each layer after the training phase. Each row displays a layer used in the CNN consecutively, the convolution parameters, the kernel size, the activation function, and the input dimensions.

We present the results of the design and implementation phases. The comparison with a well-known CNN model is essential to prove the necessity of the proposed preprocessor. In python, we first implement the LeNet-5, AlexNet, and ZFNet CNNs as the core operators of this test. Then, we script a system that generates three distinguished databases for the test. The goal is to obtain a deformed set of characters based on the testing datasets. Then, we optimize them using an optimization WNN system. In the end, the three databases’ results processed by the CNNs will show the difference in the efficiency on each dataset. Next, we show how the prediction accuracy decrease dramatically in guessing the deformed shapes. Finally, the results improve using the preprocessor’s output, without any additional training or modification in the CNNs models.

Table 2 shows the training accuracy of the CNNs on each dataset. The supervised learning approach used to train the CNNs on three samples and to test their accuracy against three testing datasets. Consequently, the networks return a high accuracy.

After finishing the training phase, each model processed both the deformed and optimized datasets and returned the expected results that highlights the advantage of the preprocessor. Table 3 shows the accuracy values of each CNN test on the databases. The good results directly decline in the first step when operating on the deformed samples. The fault gesture of the deformed shapes reflects a decline in the levels of accuracy. Finally after preprocessing, the last results reveal an improvement in accuracy with the optimized shapes, which is achieved without any CNN model modification or retraining.

| TABLE 1. | The LeNet-5 model design settings. |
|----------|---------------------------------|
| Layer Type | Output Shape | Parameters |
| Conv 2D | (28, 28, 6) | 156 |
| Pooling 2D | (14, 14, 6) | 0 |
| Conv 2D | (10, 10, 16) | 2416 |
| Pooling 2D | (5, 5, 16) | 0 |
| Conv 2D | (1, 1, 120) | 48120 |
| Flatten | 120 | 0 |
| Dense 2D | 84 | 10164 |
| Dense 2D | 10 | 850 |
| **Total Trainable Parameters:** | 61,706 |

| TABLE 2. | The CNN models training results. |
|----------|---------------------------------|
| dataset | Training Accuracy |
| Hand-Wr | 108000 | 10000 | 97.61% | 98.87% | 99.18% |
| Font | 1760 | 250 | 99.60% | 100.0% | 100.0% |
| Arabic | 9600 | 1200 | 88.83% | 93.76% | 95.75% |

VOLUME 10, 2022
TABLE 3. Shows the compilations before and after preprocessing.

| CNN         | Deformed VS Optimized | Dataset | Deformed | Optimized |
|-------------|------------------------|---------|----------|-----------|
| LeNet-5     | Hand-Written           | 10000   | 28.76%   | 93.22%    |
| AlexNet     | Hand-Written           | 10000   | 39.94%   | 95.33%    |
| ZFNet       | Hand-Written           | 10000   | 40.46%   | 94.05%    |
| LeNet-5     | Font                   | 250     | 16.40%   | 92.00%    |
| AlexNet     | Font                   | 250     | 19.20%   | 94.40%    |
| ZFNet       | Font                   | 250     | 17.20%   | 96.80%    |
| LeNet-5     | Arabic                 | 1200    | 25.00%   | 78.17%    |
| AlexNet     | Arabic                 | 1200    | 32.42%   | 82.42%    |
| ZFNet       | Arabic                 | 1200    | 35.08%   | 88.83%    |

Figure 11 illustrates multiple guessing trials by the AlexNet on the deformed Font dataset before being optimized by the preprocessor. The figure depicts a sample associated with each character. In some cases, the predictions show how 0 is confused for 9 and how 1 is for 7, which proves that the CNN is misguided in assimilating the characters. This clarifies how it is insufficient to have a clear symbol, but it should be well distributed on the input grid.

The testing dataset has a benchmark of homogeneous setting of the training images [21]. Thus, a well training exercise will lead to well scores in the testing phase. Based on the test samples, the reshaping agent generates a permutation of characters, which will reflect a drop in the CNN prediction efficiency. The same deformed dataset is then treated by the preprocessor to generate a set of rectified scripts of size 28 x 28.

Figure 12 shows the life cycle of a character in the preprocessor. The translator moves the characters and the scale resize them. While the Averager and the ReLU soften and improve the pixels. The final output is more homogeneous with the original data. Some characters, like number 1 in the figure, are already not affected by the reshaping but are slightly enhanced by the preprocessor. While other characters like 0 or 5 are resized to solve the concentration of pixels in a peripheral side of the canvas, leading to unforeseen predictions.

Of course, after all the mention results and analysis, the first thing that may jump to our minds is the concept of data augmentation. Thus we do some experiments on the data augmentation method to verify that the CNN will not converge to an optimum model resistant to input perturbation attack [13]. Thus, we select to experiment on the LeNet-5 model which already proved efficiency when data augmentation was experimented on the MNIST handwritten dataset. However, for the purpose of avoiding repetition, we select to experiment on the Font dataset and the Arabic handwritten dataset, where we expect that the selected model will converge after the training phase on each model but will fail when fed with the perturbed dataset.

Table 4 shows the results of the data augmentation method in contrast with the application of our preprocessors. Using the augmented data in both datasets, the LeNet-5 model was well trained to fulfill character classification. But, the model was not immune to character reshaping [11]. The testing results in the table of the augmented model decreased remarkably. However, when re-enforcing the LeNet-5 with our preprocessors the results of the testing phase did not decrease that must and remained acceptable.

The reason behind this reduction in performance is due to the following two reasons. First, the margin of divergence allowed in data augmentation. Usually, when applying changes to training records a small amount of change is tolerated in order to preserve the spatial distribution of distinguished features in a certain symbol or shape [3]. In consequence, any extreme change in the pattern itself will confuse the CNN model. This takes us to the second reason, the nature of the CNN model that strive to adjust their...
weights according to specific features in specific locations in order to familiarize the symbols trained for classification [1]. Figure 13 shows the difference between the locations of the original dataset specimens and the perturbed ones. The location of the feature is important in reinforcing the weights of certain neurons in the CNN layers. Thus, spatial change of the feature affected the response of the CNN model to the training stage. In conclusion, the model was case specific when learning and did not generalize its gesture of features.

![Figure 13. Shows the difference between the spatial distribution of features in original digits and in augmented ones.](image)

One of the common issues in CNN models is when the training is complete no additional information is acquired by the system [22]. Usually, in theoretical or experimental applications, the datasets are well organized in size and location. However, in real-time systems, sensors or cameras get their raw inputs directly from the surrounding [23]. Thus, some results of those systems may be mistakenly classified. Finally, may lead to misguided reactions from the machines embedding those systems [24]. Therefore, by using WNN, we aim to raise the ability of the CNN systems by a preprocessor that may improve the input form or correct any fault in objects representation [25]. The Preprocessor works to reposition the attributes of each character to levitate a sharp distribution of each pattern on a definitive space area. This network can avoid the problems caused by the location or size variance existing in the training or test datasets.

The RAM based technology can be implemented using simple micro-controllers especially for embedded systems. This allows packing graphical functions in machines for real-time operations [17]. The used techniques for image translation and scaling can be applied for complex pattern detection. The overall scope is to provide hybrid systems and approach real-time applications with minimum hardware resources. The proposed networks present an economical solution by mean of Sensors and WNN technology in compact hardware, since graphical mathematics is repetitive on sets of pixels [17].

**V. DISCUSSION**

Today, all character recognition techniques utilize CNN to identify texts. It has proven its efficiency even if applied on distributed words, without requiring any semantic structure [26]. The idea of features decomposition to simpler components is the basic of deep learning [4]. However, latest works proved that adding small perturbations to the input are sufficient to mislead its conclusions [27]. In contrast, people are naturally capable to track variations, where AI still struggles in performing equivalent tasks despite the recent advances [28]. For example, some work exhibited that a one-pixel attack performed is enough to fool a network, using adversarial methods [29]. The performed works take the adversarial concept to its limits by exploring different CNNs’ vulnerability on small-scale distortions to evaluate theirs robustness [27].

In normal approaches, rules recommend to collect homogeneous samples in order to train the best network [14]. However, in real life applications, systems are required to operate in a variable environment where everything is usually in irregular form. Therefore, embedded systems require such optimization preprocessors instead of complicated ANNs requiring a lot of resources to operate [23]. The advantage of the proposed preprocessor, with multiple image processing tasks, allows to improve inputs on site even in difficult environments where network connections are not available to reach control servers.

Optimization algorithms fulfill a stability to overcome the obstacle of data distortion [30]. The iterative mechanism of the optimizing operation focuses on modifying ANNs’ parameters to the worst-case data [31]. The Robust Optimization concept operates with a minimization-maximization alteration. Perturbed data samples processed with each weight update work to minimize the loss value [3]. Here, adversarial training plays the role of a toughness mechanism to intensify the stability of the networks. Latest results revealed that, with robust networks, the adversarial trials become harder to generate [32].

Thus, we attempt to provide the maximum integration of WNNs architectures in image transformation. Many of these require little number of resources. Furthermore, some execution can be done in series by replacing the whole neural matrix by one recurrent column. This attribute allows the proposed preprocessor to be easily integrated in embedded systems. Inspired by the deep neural nets layering approach, the proposed design provides a scalable ability to assert any sequence of image processing [6].

Figure 14 shows the accuracy levels of the LeNet-5, AlexNet, and ZFNet on the Hand-written, Font, and Arabic datasets. Each dataset has three versions, regular, deformed,
and optimized. This figure exhibits a true impression on how an excellent CNN can be fooled when attacked by the right approach. The robust results totally faded on the same but deformed dataset. However, the preprocessor rearranged the characters’ representations on the canvas space. In consequence, the same CNNs recapped all the possible optimized gestures.

We have re-introduced the WNN technology that digitizes ANNs and reduces resources requirements and computation speed [33]. In our design, WNNs provide tangible data representation and the bit-wise execution. The binary structure of the image processing matrices enables notable optimization over the dataset without suffering the course of remodeling the CNN.

In Figure 15, we report the confusion matrices of the AlexNet on the Font dataset, which have 250 images each. All three dataset versions come with the same reference values. The first matrix shows a clean distribution of gestures on the diagonal line of the table. However, the second matrix reflects a remarkable confusion in the operation of the same model on the deformed dataset. The final matrix proves the success of the preprocessor in optimizing the deformed characters. In this test, the input characters have 28 x 28 dimensions. Some of our previous designs consisted of 32 x 32 dimensions. However, we retained the standard dataset characteristics to insure a discrete result clear of any suspicion.

VI. CONCLUSION

In this research, we provide a preprocessor for CNN characters’ classification. We compare the results between multiple datasets using LeNet-5, AlexNet, and ZFNet. The test shows a decline in performance on the deformed characters. While the preprocessor recovers input shapes and returns effective improvements. Fortunately, most of the records can be restructured. However, due to major deformations, some character shapes couldn’t be well reformed.

RAM based neurons addressed several applications of characters classification. However, the design techniques of the RAM nets easily simplified our task in producing the best schematics. Those nets resulted in optimizing the digits with sharp mathematical descriptions.

For the next step, we aim to provide a hardware prototype of and embedded preprocessor. Also, we research on shapes visualization analysis to generate a fingerprint for every symbol processed by a certain application. Thus, we work on a geometrical decomposition of the fonts. Thus, the digits boundaries could have an important meaning in defining the dimensional space of every character.

ACKNOWLEDGMENT

This document is part of a project concerning multiple preprocessing applications.

REFERENCES

[1] X. Zhang, J. J. Zhao, and Y. LeCun, “Character-level convolutional networks for text classification,” in Proc. 28th Int. Conf. Neural Inf. Process. Syst., vol. 1, 2015, pp. 649–657.

[2] S. Kornblith, M. Norouzi, H. Lee, and G. Hinton, “Similarity of neural network representations revisited,” in Proc. 36th Int. Conf. Mach. Learn., vol. 97, 2019, pp. 3519–3529. [Online]. Available: http://proceedings.mlr.press/v97/kornblith19a.html

[3] Y. Bengio, I. Goodfellow, and A. Courville, “Deep learning,” Nature, vol. 521, pp. 436–444, May 2016, doi: 10.1038/nature14539.

[4] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, “Geometric deep learning: Going beyond Euclidean data,” IEEE Signal Process. Mag., vol. 34, no. 4, pp. 18–42, Jul. 2017, doi: 10.1109/MSP.2017.2693418.

[5] J. Heo, S. Joo, and T. Moon, “Fooling neural network interpretations via adversarial model manipulation,” in Proc. Adv. Neural Inf. Process. Syst., vol. 32, 2019, pp. 2925–2936. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/7e6a637f6f028b8f0a1f0f7d36ac93-Paper.pdf

[6] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. Zemel, and Y. Bengio, “Show, attend and tell: Neural image caption generation with visual attention,” in Proc. 32nd Int. Conf. Mach. Learn., vol. 37, 2015, pp. 2048–2057. [Online]. Available: http://proceedings.mlr.press/v37/xuc15.html

[7] M. A. Alcorn, Q. Li, Z. Gong, C. Wang, L. Mai, W.-S. Ku, and A. Nguyen, “Strike (with) a pose: Neural networks are easily fooled by strange poses of familiar objects,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2019, pp. 4840–4849, doi: 10.1109/CVPR.2019.00498.

[8] D. Li, D. V. Vargas, and S. Koutchi, “Universal rules for fooling deep neural networks based text classification,” in Proc. IEEE Cong. Evol. Comput. (CEC), Jun. 2019, pp. 2221–2228, doi: 10.1109/CEC.2019.8790213.

[9] B. Liang, H. Li, M. Xu, P. Bian, X. Li, and W. Shi, “Deep text classification can be fooled,” in Proc. Twenty-Seventh Int. Joint Conf. Artif. Intell., Jul. 2018, pp. 4208–4215, doi: 10.24963/ijcai.2018/585.

[10] F. M. G. França, M. De Gregorio, P. M. V. Lima, and W. R. De Oliveira, “Advances in weightless neural systems,” in Proc. 22th Eur. Symp. Artif. Neural Netw., Comput. Intell. Mach. Learn., 2014, pp. 497–504. [Online]. Available: https://www.ele.n.uci.ac.br/Proceedings/esann/esannpdf/es2014-7.pdf.
