Why drop-max is effective in making convolutional neural networks (CNNs) more robust

Min Xian, Olga Kosheleva, Martine Ceberio, and Vladik Kreinovich

Abstract While convolutional neural networks (CNNs) are very effective in image processing, they are not robust: a minor change in a few pixels can drastically change the image processing result – and thus, lead to a misclassification of the image. A recent paper has shown that CNNs can be made more robust if instead of the usual max-neurons that return the largest input, we use neurons that return the second-largest input. Such neurons are known as *drop-max* neurons. In this paper, we prove that a natural robustness requirement uniquely determines the use of drop-max neurons. We also describe what type of neurons we should use if we want to achieve a stronger robustness.

1 Formulation of the problem

Convolutional neural networks (CNNs): a brief reminder. In processing images, it turned out that convolutional neural networks – CNNs, for short – are very effective; see, e.g., [1]. A CNN is a multi-layer neural network, in which each neuron from each layer is associated with a certain point in the original image. The layers work as follows:

Min Xian
Department of Computer Science, University of Idaho, Idaho Fall, Idaho 83406, USA,
e-mail: mxian@uidaho.edu

Olga Kosheleva
Department of Teacher Education, University of Texas at El Paso, 500 W. University
El Paso, Texas 79968, USA, e-mail: olgak@utep.edu

Martine Ceberio and Vladik Kreinovich
Department of Computer Science, University of Texas at El Paso, 500 W. University
El Paso, Texas 79968, USA, e-mail: mceberio@utep.edu, vladik@utep.edu

- each neuron from the first layer – associated with a spatial location – uses, as inputs $x_1, \ldots, x_n$, intensities/colors of the original image in this location and in several nearby locations, and
- each neuron from every other layer – associated with a spatial location – uses, as inputs, outputs $x_1, \ldots, x_n$ of the neurons from the previous layer corresponding to this location and to several nearby locations.

Some layers are similar to layers of the generic neural network: they apply a non-linear function (called *activation function*) to a linear combination of the outputs of the previous layer. But the main specific feature of CNNs is that, in additional to such layers, there are also two types of layers that are specific for CNNs: linear (convolutional) layers and max-pooling (max) layers:

- in a linear layer, the output signal y is a linear combination of inputs

  $$y = w_0 + w_1 \cdot x_1 + \ldots + w_n \cdot x_n,$$

  with coefficients w_i determined during the training;
- in a max-layer, the output signal y is the maximum of all the inputs:

  $$y = \max(x_1, \ldots, x_n).$$

Problem: CNNs are not robust. One of the main problems of the CNNs is that they are not very robust: changes in a small number of pixels, changes that do not change our visual perception of the image, can lead to a drastic change in the output of the neural network and thus, to a wrong classification of the image; see, e.g., [1].

How to make CNNs more robust: an idea. To find out how to make CNNs more robust, let us analyze how changes in a small number of pixels propagate via the CNN. When this change affect an input of a neuron from a linear layer or from a layer that uses a nonlinear activation function, the inputs values are kind of averaged, so the initial effect decreases. For example, if all the weights $w_1, \ldots, w_n$ are equal and add up to 1, the effect decreases to $1/n$ of the original size. If these were the only layers, the effect of the small changed would eventually decrease and become negligible.

However, in the max-layer, if one of the changed inputs is larger than all other inputs, then the output of the neuron also changes drastically. So, to make CNN more robust, a natural idea is to focus our efforts on max-neurons.

Empirical fact. The above idea has indeed been successfully implemented: namely, it turns out that replace the maximum of all the inputs with the second largest value makes CNN more robust. This was shown in [3], on the example of the breast cancer classification.

In that paper, the neurons that return the second largest of all the inputs are called *drop-max* neurons, because they compute the maximum of what will happen if we drop the largest of the original n values.

A natural question, and what we do in this paper. A natural question is: why, out of many possible ways to make maximum more robust, drop-max leads to good results? In this paper, we provide a theoretical explanation of the drop-max's success.

2 Theoretical explanation of the drop-max's success

Analysis of the problem. A max-neuron computes the maximum of its n inputs. If one of the input values is changed, we cannot always return the maximum of the original inputs – because it is possible that exactly the largest input was changed, so based on the remaining values, we cannot determine what was the value of the original largest input. Since we cannot always produce the maximum, at least we can try to be as close to the maximum as possible, e.g., generate some value which is between the second largest and the largest – or at least between the third largest and the largest.

Let us formulate this idea in precise terms.

Notations. Following standard notations from statistics (see, e.g., [2]), let us denote, for each tuple $x_1,\ldots,x_n$, by $x_{(i)}$, the i-th element in the ordering of these elements from the smallest to the largest:

$$x_{(1)} \le x_{(2)} \le \ldots \le x_{(n)}. \tag{1}$$

In these terms, the largest element from this tuple is $x_{(n)}$, and the second largest is $x_{(n-1)}$.

Definition 1. *We say that a function $f(x_1,\ldots,x_n)$ is* robust *if for every tuple $(y_1,\ldots,y_n)$, whenever we form a new tuple $(x_1,\ldots,x_n)$ by replacing one of its elements y_i with a different number, then we will have*

$$y_{(n-2)} \le f(x_1,\ldots,x_n) \le y_{(n)}. \tag{2}$$

Proposition 1. *The only robust function $f(x_1,\ldots,x_n)$ is the function $f(x_1,\ldots,x_n) = x_{(n-1)}$ that returns the second largest element of the tuple.*

Comment. What if we want to get even closer to the maximum and require that $y_{(n-1)} \le f(x_1,\ldots,x_n) \le y_{(n)}$? This condition implies the inequality $y_{(n-2)} \le f(x_1,\ldots,x_n) \le y_{(n)}$. So we conclude, from Proposition 1, that under this condition, we will still have $f(x_1,\ldots,x_n) = x_{(n-1)}$. However, the following simple example shows that for this function $f(x_1,\ldots,x_n)$, we do not always have the desired inequality. Let us take $y_1 = 1$, $y_2 = 2$, and $y_3 = 3$. In this case,

$$y_{(1)} = y_1 = 1 < y_{(2)} = y_2 = 2 < y_{(3)} = y_3 = 3.$$

Let now change the value 3 to 0, i.e., take $(x_1,x_2,x_3) = (1,2,0)$. For this new tuple, we have:

$$x_{(1)} = 0 < x_{(2)} = 1 < x_{(3)} = 2.$$

In this case, $n = 3$, so $f(1,2,0) = x_{(2)} = 1$ which is smaller than $y_{(n-1)} = y_{(2)} = 2$.

Thus, it is not possible to require the value to be that close to the maximum. In this sense, the inequality (2) is the best we can achieve.

Proof of Proposition 1.

1°. Let us first prove that the function $f(x_1,\ldots,x_n) = x_{(n-1)}$ is robust in the sense of Definition 1.

Indeed, there are only four possibilities, and we will show that the inequality (2) holds for all four of them.

1.1°. If all three original top values $y_{(n-2)} \le y_{(n-1)} \le y_{(n)}$ remain unchanged, then we have the following four sub-options for the replaced value r:

- If $r \le y_{(n-2)}$, then in the new order, we have
$$\ldots \le r \le \ldots \le y_{(n-2)} \le y_{(n-1)} \le y_{(n)}.$$
Thus, we have $x_{(n-1)} = y_{(n-1)}$ and so, $y_{(n-2)} \le x_{(n-1)} = y_{(n-1)} \le y_{(n)}$. Hence, the inequality (2) is satisfied.
- If $y_{(n-2)} < r \le y_{(n-1)}$, then in the new order, we have
$$\ldots \le y_{(n-2)} \le r \le y_{(n-1)} \le y_{(n)}.$$
Thus, we have $x_{(n-1)} = y_{(n-1)})$ and so, $y_{(n-2)} \le x_{(n-1)} = y_{(n-1)} \le y_{(n)}$. Hence, the inequality (2) is satisfied.
- If $y_{(n-1)} < r \le y_{(n)}$, then in the new order, we have
$$\ldots \le y_{(n-2)} \le y_{(n-1)} \le r \le y_{(n)}.$$
Thus, we have $x_{(n-1)} = r$ and so, $y_{(n-2)} \le r \le y_{(n)}$. Hence, the inequality (2) is satisfied.
- Finally, if $r > y_{(n)}$, then in the new order, we have
$$\ldots \le y_{(n-2)} \le y_{(n-1)} \le y_{(n)} < r.$$
Thus, we have $x_{(n-1} = y_{(n)}$. Hence, the inequality (2) is also satisfied.

1.2°. Let us now consider the case when the original top value $y_{(n)}$ is changed. Then, we have the following three sub-options for the replacing value r:

- If $r \le y_{(n-2)}$, then in the new order, we have
$$\ldots \le r \le \ldots \le y_{(n-2)} \le y_{(n-1)}.$$
Thus, we have $x_{(n-1)} = y_{(n-2)}$. Hence, the inequality (2) is satisfied.

- If $y_{(n-2)} < r \le y_{(n-1)}$, then in the new order, we have

$$\ldots \le y_{(n-2)} < r \le y_{(n-1)}.$$

  Thus, we have $x_{(n-1)} = r$. So, $y_{(n-2)} < x_{(n-1)} = r \le y_{(n-1)}$ and since $y_{(n-1)} \le y_{(n)}$, we have $y_{(n-2)} < x_{(n-1)} \le y_{(n)}$. Hence, the inequality (2) is satisfied.
- If $r > y_{(n-1)}$, then in the new order, we have

$$\ldots \le y_{(n-2)} \le y_{(n-1)} < r.$$

  Thus, we have $y_{(n-1)} = x_{(n-1)} = y_{(n-1)}$. By the definition of the ordering, we have $y_{(n-2)} \le y_{(n-1)} = x_{(n-1)} \le y_{(n)}$. Hence, the inequality (2) is also satisfied.

1.3°. Let us now consider the case when the second largest value $y_{(n-1)}$ is changed. Then, we have the following three sub-options for the replacing value r:

- If $r \le y_{(n-2)}$, then in the new order, we have

$$\ldots \le r \le \ldots \le y_{(n-2)} \le y_{(n)}.$$

  Thus, we have $x_{(n-1)} = y_{(n-2)}$. Hence, the inequality (2) is satisfied.
- If $y_{(n-2)} < r \le y_{(n)}$, then in the new order, we have

$$\ldots \le y_{(n-2)} < r \le y_{(n)}.$$

  Thus, we have $x_{(n-1)} = r$. So, $y_{(n-1)} < x_{(n-1)} = r \le y_{(n)}$ and since $y_{(n-2)} \le y_{(n-1)}$, we have $y_{(n-2)} < x_{(n-1)} \le y_{(n)}$. Hence, the inequality (2) is satisfied.
- If $r > y_{(n)}$, then in the new order, we have

$$\ldots \le y_{(n-2)} \le y_{(n)} < r.$$

  Thus, we have $x_{(n-1)} = y_{(n)}$. Hence, the inequality (2) is also satisfied.

1.4°. Finally, let us consider the case when the third largest value $y_{(n-2)}$ is changed. Then, we have the following three sub-options for the replacing value r:

- If $r \le y_{(n-1)}$, then in the new order, we have

$$\ldots \le r \le \ldots \le y_{(n-1)} \le y_{(n)}.$$

  Thus, we have $x_{(n-1)} = y_{(n-1)}$. Hence, the inequality (2) is satisfied.
- If $y_{(n-1)} < r \le y_{(n)}$, then in the new order, we have

$$\ldots \le y_{(n-1)} < r \le y_{(n)}.$$

  Thus, we have $x_{(n-1)} = r$. So, $y_{(n-1)} < x_{(n-1)} = r \le y_{(n)}$ and since $y_{(n-2)} \le y_{(n-1)}$, we have $y_{(n-2)} < x_{(n-1)} \le y_{(n)}$. Hence, the inequality (2) is satisfied.
- If $r > y_{(n)}$, then in the new order, we have

$$\ldots \le y_{(n-1)} \le y_{(n)} < r.$$

Thus, we have $x_{(n-1)} = y_{(n)}$. Hence, the inequality (2) is also satisfied.

2°. Let us now prove that every robust function has the desired form.

2.1°. Let us first prove that for every robust function, we have $f(x_1,\ldots,x_n) \le x_{(n-1)}$.

Indeed, let us form the new tuple $(y_1,\ldots,y_n)$ by replacing its largest element $x_{(n)}$ with $x_{(1)}-1$, i.e.,

$$(y_1,\ldots,y_n) = (x_1,x_2,\ldots,x_{(n)-1},x_{(1)}-1,x_{(n)+1},\ldots,x_{n-1},x_n).$$

For this tuple:

$$y_{(1)} = x_{(1)}-1 < y_{(2)} = x_{(1)} \le y_{(3)} = x_{(2)} \le \ldots \le y_{(n)} = x_{(n-1)}.$$

The tuple $(x_1,\ldots,x_n)$ is obtained from the tuple $(y_1,\ldots,y_n)$ by changing a single element. By definition of a robust function, this implies that $f(x_1,\ldots,x_n) \le y_{(n)}$. Since $y_{(n)} = x_{(n-1)}$, we conclude that indeed $f(x_1,\ldots,x_n) \le x_{(n-1)}$.

2.2°. Let us now prove that for every robust function, we have $f(x_1,\ldots,x_n) \ge x_{(n-1)}$.

Indeed, let us form a new tuple $(y_1,\ldots,y_n)$ by replacing the smallest element $x_{(1)}$ of the original tuples with $x_{(n)}+1$, i.e.,

$$(y_1,\ldots,y_n) = (x_1,x_2,\ldots,x_{(1)-1},x_{(n)}+1,x_{(1)+1},\ldots,x_{n-1},x_n).$$

For this tuple:

$$y_{(1)} = x_{(2)} \le y_{(2)} = x_{(3)} \le \ldots \le y_{(n-2)} = x_{(n-1)} \le y_{(n-1)} = x_{(n)} < y_{(n)} = x_{(n)}+1.$$

The tuple $(x_1,\ldots,x_n)$ is obtained from the tuple $(y_1,\ldots,y_n)$ by changing a single element. By definition of a robust function, this implies that $f(x_1,\ldots,x_n) \ge y_{(n-2)}$. Since $y_{(n-2)} = x_{(n-1)}$, we conclude that indeed $f(x_1,\ldots,x_n) \ge x_{(n-1)}$.

2.3°. From Parts 2.1 and 2.2, we can conclude that $f(x_1,\ldots,x_n) = x_{(n-1)}$. The proposition is proven.

3 Auxiliary result

Discussion. In the previous section, we analyzed what happens when we change one of the inputs, but what will happen if we change $k > 1$ inputs? In this case, we have the following result.

Definition 2. *Let $k > 1$ be a natural number. We say that a function $f(x_1,\ldots,x_n)$ is k-*robust *if for every tuple $(y_1,\ldots,y_n)$, whenever we form a new tuple $(x_1,\ldots,x_n)$ by replacing k of its elements y_i with different numbers, then we will have*

$$y_{(n-(k+1))} \le f(x_1,\ldots,x_n) \le y_{(n)}. \tag{3}$$

Proposition 2. *For every $k > 1$, the only k-robust function $f(x_1,\ldots,x_n)$ is the function $f(x_1,\ldots,x_n) = x_{(n-k)}$.*

Comment. Similarly to what we did after Proposition 1, we can show that this is the best possible result: in general, it is not possible to get closer to the maximum. To be more precise, it is not possible to have a stronger inequality $y_{(n-k)} \le f(x_1,\ldots,x_n) \le y_{(n)}$. Indeed, let us take $n = 2k+1$ and $y_i = i$ for all i from 1 to n, and let us replace the top k values with 0s, i.e., let us have

$$(x_1,\ldots,x_n) = (y_1,\ldots,y_{k+1},0,\ldots,0) = (1,\ldots,k+1,0,\ldots,0).$$

In this case,

$$x_{(1)} = \ldots = x_{(k)} = 0 < x_{(k+1)} = 1 < x_{(k+2)} = 2 < \ldots < x_{(2k+1)} = k+1,$$

so $f(x_1,\ldots,x_n) = x_{(n-k)} = x_{(k+1)} = 1$, but $y_{(n-k)} = y_{(k+1)} = k+1 > 1$, so the inequality $y_{(n-k)} \le f(x_1,\ldots,x_n)$ is not satisfied.

Proof of Proposition 2. This proof is similar to Proposition 1, with the following two main differences:

- To prove that $f(x_1,\ldots,x_n) \le y_{(n)}$, we replace $x_{(n)}$ with $y_{(1)} - 1$, $x_{(n-1)}$ with $y_{(1)} - 2$, …, $x_{(n-i)}$ with $y_{(1)} - (i+1)$, …, and $x_{(n-(k-1))}$ with $y_{(1)} - k$. In this case, $y_{(n)} = x_{(n-k)}$, so the k-robustness condition $f(x_1,\ldots,x_n) \le y_{(n)}$ implies that
$$f(x_1,\ldots,x_n) \le x_{(n-k)}.$$
- To prove that $f(x_1,\ldots,x_n) \ge y_{(n)}$, we replace $x_{(1)}$ with $y_{(n)} + 1$, $x_{(2)}$ with $y_{(n)} + 2$, …, $x_{(i)}$ with $y_{(n)} + i$, …, and $x_{(k)}$ with $y_{(n)} + k$. In this case, $y_{(n-(k+1))} = x_{(n-k)}$, so the k-robustness condition $f(x_1,\ldots,x_n) \ge y_{(n-(k+1))}$ implies that
$$f(x_1,\ldots,x_n) \ge x_{(n-k)}.$$

From these two inequalities, we conclude that indeed $f(x_1,\ldots,x_n) = x_{(n-k)}$. The proposition is proven.

Acknowledgments

This work was supported by the AT&T Fellowship in Information Technology, by the Institute for Risk and Reliability, Leibniz Universitaet Hannover, Germany, by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) Focus Program SPP 100+ 2388, Grant Nr. 501624329, by the European Union under the project ROBOPROX (No. CZ.02.01.01/00/22 008/0004590), by the Center of Excellence in Econometrics, Faculty of Economics, Chiang Mai University, Thailand, by the Ho Chi Minh City University of Banking, Vietnam, and by Thang Long University, Hanoi, Vietnam.

References

1. I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, Cambridge, Massachusetts, 2016.
2. D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*, Chapman and Hall/CRC, Boca Raton, Florida, 2011.
3. S. Sun, M. Xian, A. Vakanski, and H. Ghanem, "MIRST-DM: Multi-Instant RST with Drop-Max Layer for Robust Classification of Breast Cancer", *Proceedings of the 25th International Conference on Medical Image Computing and Computer Assisted Intervention MICCAI 2022, Singapore, September 18–22, 2022*, Springer Lecture Notes in Computer Science, Vol. 13434, 2022, ISBN 978-3-031-16439-2, doi 10.1007/978-3-031-16440-8_39