An analysis of ConformalLayers' robustness to corruptions in natural images

Eduardo Vera Sousa[a,*], Cristina Nader Vasconcelos[a], Leandro A. F. Fernandes[a]

[a] *Instituto de Computação, Universidade Federal Fluminense (UFF), 24210-240, Niterói-RJ, Brazil*

ABSTRACT

ConformalLayers are sequential Convolutional Neural Networks (CNNs) that use activation functions defined as geometric operations in the conformal model for Euclidean geometry. Such construction turns the layers of sequential CNNs associative, leading to a significant reduction in the use of computing resources at inference time. After the layer's association, both the processing time and memory used per batch entry become independent (i.e., constant) of the network's depth. They depend only on the size of the input and output. The cost of conventional sequential CNNs, on the other hand, presents linear growth. This paper evaluates the robustness of the classification of ConformalLayers-based CNNs against different kinds of corruptions typically found in natural images. Our results show that the mean top-1 error rates of vanilla CNNs are smaller than ConformalLayer-based CNNs on clean images. Still, our approach outperforms other optimization techniques based on network quantization, and the relative difference to vanilla networks tends to reduce in the presence of image corruptions. Furthermore, we show that processing time and CO_2 emission rates are much lower for ConformalLayer-based CNNs with a depth greater than seven and two, respectively.

1. Introduction

The pervasive usage of neural networks in our day-to-day life has significantly required improvement concerning the robustness to noise, blur, weather conditions, and lossy compression. Factors that lead to corrupted images impact the accuracy of neural networks in critical tasks like object detection for autonomous car driving (Milyaev and Laptev, 2017) and facial recognition for surveillance (Reina et al., 2021). Unfortunately, adopting complex models to increase robustness to corruptions can also increase energy consumption and CO_2 emissions.

The associative design of ConformalLayers networks developed in our previous work (Sousa et al., 2021) is suitable for the proposition of sequential network models to be deployed in edge devices, like autonomous cars. The most exciting features of ConformalLayers are their low memory footprint and reduced inference time. In our previous paper, we show

*Corresponding author

e-mail: eduardovera@ic.uff.br (Eduardo Vera Sousa), crisnv@ic.uff.br (Cristina Nader Vasconcelos), laffernandes@ic.uff.br (Leandro A. F. Fernandes)

that the computing resources used by the model at inference time are constant regardless of the depth of the neural network.

In this work, we present an analysis of the robustness of ConformalLayers-based networks to common corruptions found in natural images, including, but not limited to, variations of brightness, contrast, compression, and several cases of noise and blurring. Specifically, we use the CIFAR-10-C benchmark (Hendrycks and Dietterich, 2019) to assess the robustness of classification networks. Additionally, we compare ConformalLayers-based and conventional CNNs with varying depth concerning inference time and CO_2 emission. Our results show that networks built using ConformalLayers, despite having lower top-1 accuracy than the conventional implementation of the same architectures, present less relative degradation in the classification accuracy of corrupted images, outperform neural network quantization approaches, and have advantages on inference time and CO_2 emission. Regarding processing time, the new implementation of ConformalLayers proved to be more advantageous on networks with more than eight layers and less polluting on CNNs with more than two layers. In the previous work, we observed advantages from the ninth layer.

Our main contributions are: (1) The evaluation of the robustness of ConformalLayers concerning common corruptions in natural images; and (2) The analysis of inference time and energy consumption on networks with varying depths.

2. Related Work

This section presents a brief literature review on strategies for improving the robustness of neural networks against noise.

Adversarial Training. Szegedy et al. (2014) presented that small data noise may trick neural network models, leading to misclassifications even in models with high accuracy. Szegedy's et al. ideas led to using adversarial training to improve the robustness of the models. Kireev et al. (2021) demonstrated that using a selected parameter in an adversarial training context could provide a baseline against common corruptions and improve the model robustness. Addepalli et al. (2021) presented a training-time approach for adversarial training, which can outperform state-of-the-art techniques.

Data Augmentation. Yin et al. (2019) performed an analysis of the model robustness improvement obtained by data augmentation and adversarial training-based approaches under a frequency domain perspective. This allowed them to check which frequencies the robustness reduction occurred, proposing data augmentation driven towards mitigating this issue. Rusak et al. (2020) and Lopes et al. (2020), on the other hand, focused on a specific kind of noise with guided data augmentation. The former presented an approach for robustness improvement of neural network models that include small portions of Gaussian and speckle noise in the training set. The latter proposed a data augmentation scheme to

deal with the trade-off between accuracy and robustness by adding Gaussian noise to randomly selected patches in the training set. The data augmentation approach proposed by Hendrycks et al. (2020) is guided by the dataset distribution to enhance the robustness to common corruptions.

The idea of changing the policies for data augmentation was introduced by Lyzhov et al. (2020), with the Greedy Policy Search. Their approach uses the test dataset as a reference to learn a policy for data augmentation that increases the robustness of the model in the following train steps.

Model Tuning. Lee et al. (2020) achieved good results related to the model accuracy and robustness to common corruptions by implementing a model ensemble-based approach. The authors claim that there is only a small drawback on inference throughput despite the usage of such model aggregation. Benz et al. (2021) relied on batch normalization adjustments to fix the data distribution and improve the model's resiliency, also considering the corrupted datasets.

Corruption Benchmarks. Hendrycks and Dietterich (2019) provided a benchmark to compare multiple neural network architectures concerning the robustness to the common types of corruptions in natural images. In this work, we use the Hendrycks and Dietterich's dataset in our experiments.

3. ConformalLayers

ConformalLayers is a new paradigm presented by Sousa et al. (2021) for building sequential Convolutional Neural Networks (CNNs) using layers in which operations are associative. By *associativity* we refer to the mathematical property of a binary operation $\circ$ on a set $\mathcal{S}$ to satisfy the associative law:

$$(x \circ y) \circ z = x \circ (y \circ z) \text{ for all } x, y, z \in \mathcal{S}. \tag{1}$$

The central idea of ConformalLayers is: given that the layers are associative, then after training, they can be combined, simplifying the operation to be applied to the data at inference time. Such simplification leads to fewer arithmetic operations and reduces the number of intermediate feature maps produced by conventional processing, significantly reducing the processing time, energy consumption, and memory footprint.

Layers typically adopted in CNNs, such as max-pooling and popular activation functions such as ReLU, cannot be associated with convolutional layers or linear (*i.e.,* fully connected) layers, among others.

In Section 3.1, we present the associative activation function proposed by Sousa et al. (2021). Section 3.2 shows how ConformalLayers can be defined through products and sums of tensors, exploring the associativity of those operations. We discuss the test-bed implementation of ConfomalLayers in Section 3.3. Table 1 presents the notation conventions used in this section.

3.1. ReSPro

ReSPro stands for Spherical Reflection, Scaling, and Projection. Those are the operations applied to the point x encoding the input feature map of the ReSPro activation function, producing a new point y that corresponds to the resulting feature map. Each coordinate of these points corresponds to a coefficient of the data. For example, a feature map with 10×10 pixels and 3 channels is a point $x = (x_1, x_2, \cdots, x_d)$ with $d = 300$ coordinates, which after being transformed by ReSPro maps to $y = (y_1, y_2, \cdots, y_d)$. The x-to-y mapping involves one non-linear and two linear transformations applied to x. Fig. 1 illustrates the mapping performed by ReSPro assuming $d = 1$.

Let z and x be two points lying on the e_d axis, distant, respectively, α and δ units from the origin, for $0 \leq \delta \leq \alpha$. In Fig. 1, we deliberately added the extra dimension e_{d+1} to the Cartesian space and placed the hypersphere S (a circle, in this example) with radius α and center $c = (0, \alpha)$. In this new space, $z = (z_d, z_{d+1}) = (-\alpha, 0)$ and $x = (x_d, x_{d+1}) = (\delta, 0)$. The addition of the dimension e_{d+1} to the Cartesian coordinate system is key for defining the non-linear transformation in ReSPro as the spherical reflection of points on S. The spherical reflection maps z and x to, respectively, z' and x'.

Spherical reflection causes points outside the hyperspherical mirror to produce images inside the mirror. The distance of the imaged point to the center of the hypersphere decreases non-linearly as the distance of the original point to the center increases. Notice in Fig. 1 that the distance between points z' and c is smaller than the distance between x' and c since z is more distant from c than x. At the limit, points at infinity map to c, while points on the hypersphere remain unchanged. By construction, we do not have to care about the reflection of points inside the hyperspherical mirror because S is tangent to the space spanned by $\{e_1, e_2, \cdots, e_d\}$, and all input points lie in this space (*i.e.,* the coordinate for e_{d+1} is zero).

Two linear transformations are applied after spherical reflection. The first is isotropic scaling by a factor of $2/\alpha$, which maps z' and x' to z'' and x'', respectively. The second transformation is the orthogonal projection to the original d-dimensional Cartesian space. As can be seen in Fig. 1, z'' projects to $(-1, 0)$ while x'' projects to $y = (y_d, 0)$. The

Table 1: Notation conventions.

Symbol	Description
x, y, p	Points in Cartesian d-dimensional space.
x_i, y_i, p_i	The i-th coordinate of points x, y, and p.
X, Y, P	Vectors encoding points x, y, and p in the $(d+3)$-dimensional space of the conformal model of geometry.
x'_o, y'_o	Homogeneous coefficient of vectors X and Y, such that $x'_o \neq 0$ and $y'_o \neq 0$.
x'_i, y'_i	The i-th coefficient of vectors X and Y, such that $x_i = x'_i/x'_o$ and $y_i = y'_i/y'_o$.
α	Radius of the hypersphere used in the ReSPro.
F_M	Diagonal matrix of size $(d+1) \times (d+1)$ used in the tensor representation of the ReSPro.
F_T	Rank-3 tensor of size $(d+1) \times (d+1) \times (d+1)$ used in the tensor representation of the ReSPro.
U	Matrix encoding a sequence of linear operations.
$\mathcal{L}$	The conformal layer function.

reasoning for performing scaling followed by projection is to map points that are α units away from the origin of the Cartesian space to points 1 unit away from the origin. The projection also makes the coordinate associated with the extra dimension e_{d+1} equal to zero, which can be removed from the resulting point since it is constant. Formally, ReSPro is a mapping

$$f : x \to y, \text{ subject to } \|x\|_2 \in [0, \alpha] \text{ and } \|y\|_2 \in [0, 1]\,, \tag{2}$$

where $x, y \in \mathbb{R}^d$ are, respectively, the input and output data represented as points in Cartesian d-dimensional space, $\|\cdot\|_2$ denotes the L^2-norm, and α is the radius of the hypersphere used in the reflection.

ReSPro is non-linear, differentiable, invertible, and associative to the linear functions typically used in CNNs. We use the geometric algebra of the conformal model for Euclidean geometry (Dorst et al., 2010) to perform spherical reflection as orthogonal transformation (hence, a linear transformation) and combine it with isotropic scaling and orthogonal projection. The representational space of the conformal model has two extra dimensions and a degenerate metric (Dorst et al., 2010). By definition, we also include one extra dimension (e_{d+1}) to the Cartesian space with basis vectors $\{e_1, e_2, \cdots, e_d\}$. Hence, the $(d+1)$-dimensional Cartesian space is embedded in a $(d+3)$-dimensional space with basis vectors $\{e_1, e_2, \cdots, e_{d+1}, e_o, e_\infty\}$. The extra dimensions e_o and e_∞ are geometrically interpretable in the conformal model as the point at the origin and the point at infinity, respectively. In the conformal model, a finite point with coordinates $p = (p_1, p_2, \cdots, p_d, p_{d+1})$ is encoded by the $(d+3)$-dimensional vector:

$$P = \gamma \left(p_1, p_2, \cdots, p_d, p_{d+1}, 1, -\frac{1}{2} \sum_{i=1}^{d+1} (p_i)^2 \right)^\intercal , \tag{3}$$

where $^\intercal$ denotes matrix transposition and the scalar $\gamma \neq 0$ does not change the practical interpretation of P as the point p. Please refer to Dorst et al. (2010) for a comprehensive explanation of how to represent geometric entities and Euclidean transformations in the conformal model of geometry.

The algebraic manipulation that takes the ReSPro function from its formulation in geometric algebra to tensor algebra is quite involved, and it is presented by Sousa et al. (2021). Here, it is sufficient to show that, in tensor form, a d-dimensional point $x = (x_1, \cdots, x_d)$ encoding input data will be represented by the $(d+1)$-dimensional vector:

$$X = (x'_1, \cdots, x'_d, x'_o)^\intercal , \tag{4}$$

such that $x_i = x'_i / x'_o$ and $x'_o \neq 0$, for $i \in \{1, 2, \cdots, d\}$. Vector X is mapped to Y by the ReSPro function as:

$$Y = (y'_1, \cdots, y'_d, y'_o)^\intercal = \left(x'_1, \cdots, x'_d, \frac{\alpha}{2} x'_o + \frac{1}{2\alpha} \sum_{i=1}^{d} (x'_i)^2 \right)^\intercal \tag{5}$$

such that $y_i = y'_i / y'_o$ and $y'_o \neq 0$, for $i \in \{1, 2, \cdots, d\}$.

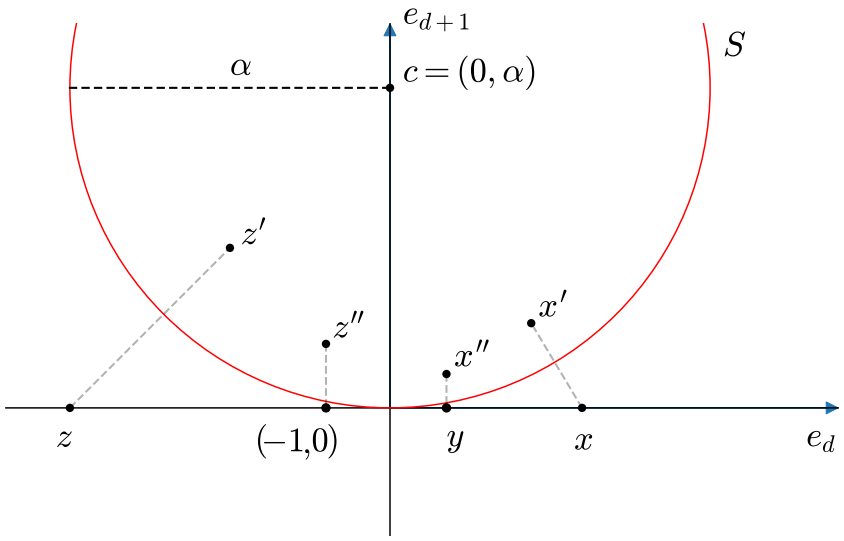

Fig. 1: Mapping performed by the ReSPro function.

In (3), P is a vector that encodes any finite point p according to the convention of the conformal model of geometry. Like P, X and Y are vectors that follow the same convention. However, these are, respectively, the input and output of the ReSPro function. Furthermore, in (4) and (5), X and Y are simplified version of P. They do not include the coefficient related to e_{d+1}, since this coefficient is always zero at the beginning and the end of the mapping, nor the coefficient related to e_∞, because it can be computed from the other coefficient. The x'_o and y'_o coefficients act as homogeneous coordinates. By dividing X by x'_o and Y by y'_o, one obtains the coordinates as points x and y, respectively.

Activation functions can affect the learning rate and training time, or lead to vanishing/exploding gradients. We did not notice ReSPro suffering from these issues during our experiments. We believe that the reason is that ReSPro behaves like an element-wise activation followed by normalization, which would give some stability to the training.

3.2. Tensor Representation of ConformalLayers

The tensor representation of (5) is:

$$Y = (y'_1, \cdots, y'_d, y'_o)^\intercal = (F_M + F_T X)\, X, \tag{6}$$

where

$$F_M = \begin{pmatrix} 1 & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 1 & 0 \\ 0 & \cdots & 0 & \frac{\alpha}{2} \end{pmatrix} \text{ and } F_T X = \begin{pmatrix} 0 & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & 0 \\ \frac{x'_1}{2\alpha} & \cdots & \frac{x'_d}{2\alpha} & 0 \end{pmatrix}.$$

F_M is a $(d+1)\times(d+1)$ diagonal matrix, and F_T is a rank-3 tensor of size $(d+1)\times(d+1)\times(d+1)$ filled with zeros, except for the slice at the bottom, which is the diagonal matrix:

$$F_{T[d+1]} = \begin{pmatrix} \frac{1}{2\alpha} & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & \frac{1}{2\alpha} & 0 \\ 0 & \cdots & 0 & 0 \end{pmatrix}.$$

Recall that X in (6) represents a finite point $x \in \mathbb{R}^d$, whose coordinates are the coefficients of some feature map produced by some CNN layer. It is well known that linear functions are typically used in layers and their configurations can be written in matrix form and applied to vectors by matrix multiplication. For instance, Toeplitz matrices encode n-dimensional discrete-time convolutions and can be modified to encode valid cross-correlation (Goodfellow et al., 2016); average pooling is the mean filter (Gonzalez and Woods, 2008), a particular case of convolution with constant weights; and configurations such as padding, dilation, and stride can be encoded by matrices composed of zeros and ones (see Sousa et al. (2021) for details). By writing these and other operations in the matrix form, the associativity of the matrix product allows the composition of operations to produce matrices U which, when multiplied by a vector X, produce the vector $Z = UX$ representing the output of a sequence of linear layers. By replacing X in (6) by Z and combining U with F_M and F_T, we write:

$$\begin{aligned} Y = \mathcal{L}(X) &= \left(F_M + F_T\left(UX\right)\right)\left(UX\right) \\ &= \left(F_M U + (U^\intercal F_T^\intercal U)^\intercal X\right) X, \end{aligned} \tag{7}$$

where $\mathcal{L}$ is *the conformal layer function*, a sequence of linear operations applied to X, followed by the application of the ReSPro activation function. Here, $^\intercal$ denotes the transposition of the first two dimensions of tensors and matrix transposition.

A sequence of k conformal layers applied to X is written as:

$$Y^{(k)} = \mathcal{L}^{(k)}(\mathcal{L}^{(k-1)}(\mathcal{L}^{(k-2)}(\cdots))), \tag{8}$$

where $\mathcal{L}^{(0)} = X$. In $\mathcal{L}^{(l)}$, the upper index identifies the l-th conformal layer, $X = \left(x'_1, \cdots, x'_{d_{\text{in}}}, x'_o\right)^\intercal$ encodes the input data, and $Y^{(k)} = \left(y'^{(k)}_1, \cdots, y'^{(k)}_{d_{\text{out}}}, y'^{(k)}_o\right)^\intercal$ encodes the output of the k-th layer, whose actual coefficients as point coordinates can be computed as $y_j = y'^{(k)}_j / y'^{(k)}_o$, for $j = \{1, 2, \cdots, d^{(k)}_{\text{out}}\}$.

We expand (8) to model a sequence of k conformal layers applied to X using tensor form:

$$Y^{(k)} = \left(L_M^{(k)} + L_T^{(k)} X\right) X, \tag{9}$$

where

$$L_M^{(k)} = F_M^{(k)} U^{(k)} F_M^{(k-1)} U^{(k-1)} \cdots F_M^{(1)} U^{(1)} \tag{10}$$

is a $(d_{\text{out}}^{(k)} + 1) \times (d_{\text{in}} + 1)$ sparse matrix, and

$$L_T^{(k)} = \sum_{l=1}^{k} \left(F_M^{(k)} U^{(k)} F_M^{(k-1)} U^{(k-1)} \cdots F_M^{(l+1)} U^{(l+1)} \right) \left(U^{(1)\intercal} U^{(2)\intercal} \cdots U^{(l)\intercal} F_T^{(l)\intercal} U^{(l)} \cdots U^{(2)} U^{(1)} \right)^{\intercal} \quad (11)$$

is a $(d_{\text{out}}^{(k)} + 1) \times (d_{\text{in}} + 1) \times (d_{\text{in}} + 1)$ tensor filled with zeros, except for the slice at the bottom, which is a sparse $(d_{\text{in}} + 1) \times (d_{\text{in}} + 1)$ matrix. The Supplementary Material of Sousa et al. (2021) shows how to turn (8) into (9).

Notice that the $L_M^{(k)}$ and $L_T^{(k)}$ components of (9) do not depend on the input data X, and they encode a complete sequence of k conformal layers. Therefore, after the weights of the CNN be adjusted through the training process, $L_M^{(k)}$ and $L_T^{(k)}$ can be computed once by associativity using (10) and (11), and applied afterward to any X to perform inference.

3.3. Implementation of ConformalLayers

The source code of the ConformalLayers is available at `https://github.com/Prograf-UFF/ConformalLayers/`. We have implemented it as an `nn.Module` of PyTorch 1.9. The `cl.ConformalLayers` module behaves like an `nn.Sequential` module, and its current version accepts submodules that mimic the behavior of `nn.Conv1d`, `nn.Conv2d`, `nn.Conv3d`, `nn.AvgPool1d`, `nn.AvgPool2d`, `nn.AvgPool3d`, `nn.Dropout`, `nn.Flatten`, and `nn.Identity`, in addition to the `cl.ReSPro` module. The configuration of existing modules includes stride, zero padding, dilation, and channel grouping whenever necessary. We plan to include complementary modules such as `nn.ConvTranspose`*`N`*`d` and `nn.Linear`, and arguments like `bias` in future versions of our library.

During the training process, the `cl.ConformalLayers` module uses the native PyTorch implementation of supported submodules to compose the network. The only exception is the `cl.ReSPro` module, which is implemented according to (5), and the computation of the coefficient $y_o'^{(k)}$ in all submodules. Once the training process is completed, the `cl.ConformalLayers` module builds an internal cache containing the sparse matrix $L_M^{(k)}$ (10) and the sparse matrix representing the slice at the bottom of $L_T^{(k)}$ (11). This cache has to be updated only if the user decides to retrain the model. Due to the sparse nature of $L_M^{(k)}$ and $L_T^{(k)}$, the expression in (9) is evaluated using only two sparse matrix-vector multiplications, one dot product of vectors, one addition, and one multiplication. As the value of $y_o'^{(k)}$ can be different from 1, the result of (9) needs to be divided by $y_o'^{(k)}$ to obtain the correct result.

The calculation of sparse tensors $U^{(l)}$ used in the computation of $L_M^{(k)}$ and $L_T^{(k)}$ is implemented using the Minkowski Engine 0.5.4 (Choy et al., 2019), as PyTorch 1.9 does not implement most of its modules to process sparse tensors.

The module `cl.ReSPro` accepts the user to explicitly enter the value of its $\alpha^{(l)}$ argument while defining the network

architecture or leave it for the `cl.ConformalLayers` module estimate the $\alpha^{(l)}$ value. For doing so, we need to keep the tracking of the maximum distance from the origin the point interpretation of the result of the linear operations in the l-th conformal layer may have. Before applying the $U^{(l)}$ matrix (which encodes the linear operations), it is reasonable to assume that the L^2-norm of the layer's input point is 1, as long as we set the x'_o coordinate of the input data to the Euclidean distance of x to the origin. By doing so, all input vector X will encode a point 1 unit away from the origin, and, by definition, the vector resulting from the application of ReSPro in the previous layer (*i.e.,* $Y^{(l-1)}$, for $l > 1$) encodes points distant up to 1 unit from the origin. But after applying $U^{(l)}$, the maximum distance of the point given to the `cl.ReSPro` module may change from 1 to any value, depending on the composition of $U^{(l)}$. Therefore, our implementation progressively updates $\alpha^{(l)}$ to the upper limit for the L^2-norm that each operation may produce.

For `nn.AvgPool`N`d`, `nn.Dropout`, and `nn.Flatten`, the upper limit for the distance of the resulting point to the origin is equal to the upper limit given as input. For `nn.Conv`N`d` without bias, Young's inequality (Young, 1912) defines the boundaries of the convolution operator $*$ as:

$$\|g * h\|_r \ \leq \ \|g\|_p \ \|h\|_q, \text{ subject to } \frac{1}{p} + \frac{1}{q} = \frac{1}{r} + 1, \tag{12}$$

where g and h denote two discrete signals, and $\|\cdot\|_t$ denotes the L^t-norm. In our case, we use $p = 2$, $q = 1$, and $r = 2$ for estimating the upper bound for the L^2-norm of the output.

The usage of ConformalLayers is transparent to PyTorch users, as can be seen in Fig. 2. We were careful to keep the same signature for the supported modules.

4. Materials and Methods

In this section, we characterize the environment and the metrics used in our experiments for the assessment of two aspects of ConformalLayers: (i) processing time and energy consumption, here presented as the amount of CO_2 emitted at inference; and (ii) robustness to common corruptions in natural images.

Hardware. We run our experiments in an Intel Xeon 8160 CPU with 2.10GHz, 376Gb of RAM, and 4 GPUs NVIDIA Tesla V100 with 32Gb of memory each. We have executed all experiments inside a Docker container with RAM limited to 32Gb and the number of visible GPUs set to one.

Performance Measurement. For measuring inference time, we have performed forward propagation while processing batches of 64 images in DkNet and DkNetCL networks (Sousa et al., 2021). These networks have k layers that receive and produce feature maps with the same resolution (see Fig. 3). They were specially designed for measuring the

```
import torch
import cl

class SomeNet(torch.nn.Module):
    def __init__(self):
        super(SomeNet, self).__init__()
        self.features = torch.nn.Sequential(
            torch.nn.Conv2d(3, 6, kernel_size=5),
            torch.nn.ReLU(),
            torch.nn.AvgPool2d(kernel_size=2, stride=2),
            torch.nn.Conv2d(6, 16, kernel_size=5),
            torch.nn.ReLU(),
            torch.nn.AvgPool2d(kernel_size=2, stride=2),
        )
        self.classifier = ...
    ...

class SomeCLNet(torch.nn.Module):
    def __init__(self):
        super(SomeCLNet, self).__init__()
        self.features = cl.ConformalLayers(
            cl.Conv2d(3, 6, kernel_size=5),
            cl.ReSPro(),
            cl.AvgPool2d(kernel_size=2, stride=2),
            cl.Conv2d(6, 16, kernel_size=5),
            cl.ReSPro(),
            cl.AvgPool2d(kernel_size=2, stride=2),
        )
        self.classifier = ...
    ...
```

Fig. 2: Code snippet comparing the use of conventional PyTorch modules (top) and ConformalLayers modules (bottom).

$k \times$

DkNet	**DkNetCL**
Input (32×32 RGB image)	
Conv o=32 k=3 p=1 b=on ReLU	Conv o=32 k=3 p=1 b=off ReSPro
Flatten	
Linear o=10 b=on	

BaseReLUNet	**BaseReSProNet**
Input (32×32 RGB image)	
Conv o=32 k=3 b=on ReLU	Conv o=32 k=3 b=off ReSPro
AvgPool k=2 s=2	
Flatten	
Linear o=10 b=on	

LeNet	**LeNetCL**	**LeNetBinary**	**LeNetMasked**
Input (32×32 RGB image)			
Conv o=6 k=5 b=on ReLU	Conv o=6 k=5 b=off ReSPro	Conv o=6 k=5 b=on HardTanh	Conv o=6 k=5 b=on ReLU
AvgPool k=2 s=2			
Conv o=16 k=5 b=on ReLU	Conv o=16 k=5 b=off ReSPro	Conv o=16 k=5 b=on HardTanh	Conv o=16 k=5 b=on ReLU
AvgPool k=2 s=2			
Flatten			
Linear o=120 b=on ReLU	Linear o=120 b=on ReLU	Linear o=120 b=on HardTanh	Linear o=120 b=on ReLU
Linear o=84 b=on ReLU	Linear o=84 b=on ReLU	Linear o=84 b=on HardTanh	Linear o=84 b=on ReLU
Linear o=10 b=on			

Fig. 3: Networks' configuration. Here, `o` is the number of output channels, `k` is kernel size, `p` denotes padding, `s` is stride, and `b` indicates the use of bias.

computing resources required concerning the depth of the network since the only changing variable here is k. As we increase the k-value, we can assess the impact of the depth on inference time.

We also assess the CO_2 emission rate of DkNet and DkNetCL networks during the inference step. For that, we have relied on the CodeCarbon toolset (Schmidt et al., 2021).

The classification accuracy is not important for the experiment involving performance measurement. Therefore, we have not trained those models, and we have used uniformly distributed random data as input. The analyzes consider

the average of measurements over 50 executions of each network. We use Welch's t-test (Welch, 1947) to compare mean values. It does not assume equal population variance to test the null hypothesis that two populations have equal means. The alternative hypothesis is that the means of the distributions are unequal. We have assumed a significance level $\alpha = 0.05$. Therefore, we reject the null hypothesis if the p-value is less or equal to 0.05.

Datasets. For the robustness assessment, we rely on the well-known corruptions dataset CIFAR-10-C, provided by Hendrycks and Dietterich (2019), which consists of 19 different types of corruptions applied to natural images originally from the CIFAR-10 dataset (Krizhevsky, 2009). Each type of corruption has 5 different levels of severity. The CIFAR-10-C dataset includes 10,000 test images for each combination of corruption type and severity level, while CIFAR-10 has a training set of 50,000 images and a test set of 10,000 images.

Robustness Assessment. The metrics used for robustness assessment are the Mean Corruption Error (mCE), which measures the error by aggregating the top-1 error rates under different levels of corruption severity, and Relative mCE, which measures how the classification degrades in the presence of corruptions. They were proposed together with the CIFAR-10-C dataset by Hendrycks and Dietterich (2019).

To calculate the mCE, let $E_{s,c}^{f}$ denote the error rate obtained by the network f associated to the corruption c at severity level s. The Corruption Error value is:

$$CE_{c}^{f} = \left(\sum_{s=1}^{5} E_{s,c}^{f}\right) \Bigg/ \left(\sum_{s=1}^{5} E_{s,c}^{\text{Baseline}}\right). \tag{13}$$

In (13), $E_{s,c}^{\text{Baseline}}$ is a normalization term considering the error in a baseline architecture. Hendrycks et al. (2020) used AlexNet (Krizhevsky et al., 2012) as baseline. As they do not impose restrictions on which network should be used as baseline, in this work, we consider LeNet-5 (Lecun et al., 1998) and BaseReLUNet (Sousa et al., 2021) for normalization because those are the vanilla networks. The average of the Corruption Error values is the mCE metric:

$$mCE^{f} = \frac{1}{19}\sum_{c=1}^{19} CE_{c}^{f}. \tag{14}$$

The Relative CE is calculated as:

$$RCE_{c}^{f} = \left(\sum_{s=1}^{5} E_{s,c}^{f} - E_{\text{Clean}}^{f}\right) \Bigg/ \left(\sum_{s=1}^{5} E_{s,c}^{\text{Baseline}} - E_{\text{Clean}}^{\text{Baseline}}\right), \tag{15}$$

where E_{Clean}^{f} refers to the error associated with an architecture f in a dataset without corruptions, *i.e.*, the test subset of the CIFAR-10 dataset. Thus, the Relative mCE is given by:

$$\text{Relative } mCE^{f} = \frac{1}{19}\sum_{c=1}^{19} RCE_{c}^{f}. \tag{16}$$

Fig. 3 presents the configuration of the networks used for robustness assessment. We designed the BaseReLUNet and BaseReSProNet architectures to analyze the influence of ReSPro on the classification problem considering a straightforward network with a single layer. We also compare the vanilla LeNet against the ConformalLayers-based version (LeNetCL) and two LeNet-based networks that apply compression strategies from the literature. The LeNetBinary uses the concepts presented by Hubara et al. (2016), which described an approach to mitigate resource and energy usage of neural networks by binarizing the network's weights. The LeNetMasked uses the masking criteria presented by Zhou et al. (2019) to assess which weights should be pruned to find a simpler network according to lottery ticket hypothesis. The reason for using LeNet-based models for comparison is that LeNet is a well-known architecture, as it has already been extensively studied in the literature.

For each CNN, we've performed hyperparameter optimization via a Bayesian approach (Bergstra et al., 2013) assuming validation accuracy as metric and Hyperband (Li et al., 2017) as stopping criteria, with arguments `min_iter` $= 10$ and $\eta = 3$. Please refer to the Supplementary Material for the hyperparameter search space and the hyperparameter values selected for each CNN. In all experiments, we set the ConformalLayers to automatically estimate the α values used by the ReSPro activation functions. The networks were trained with a random and balanced subset of 40,000 images and validated with the remaining 10,000 images from the CIFAR-10 training set. We have compared the mean top-1 error rates using Welch's t-test (Welch, 1947) with $\alpha = 0.05$.

5. Experiments and Results

In the first analysis, we assess the mean execution time and mean CO_2 emission rate of CNNs with varying depth. Fig. 4 summarizes the performance of DkNet, a network built in a non-associative way using ReLU, and DkNetCL, a network built with ConformalLayers, and hence, using ReSPro. The first notable observation about processing time (Fig. 4, top) reveals that DkNetCL presents constant behavior, regardless of the number of layers. DkNet, on the other hand, presents monotonically non-decreasing cost, *i.e.,* as we increase the depth of the network the inference time also increases. Notice that DkNet is almost $3\times$ faster than DkNetCL for $k = 1$. As the neural networks become deeper ($k \geq 8$), however, DkNetCL becomes more advantageous. For $k = 7$, the p-value is 0.49. For all other cases, it is less than 0.05. The implementation of the `cl.ConformalLayers` module used by Sousa et al. (2021) includes steps that were simplified in the current version, and the PyTorch version is also different. These modifications resulted in a performance improvement. Sousa et al. (2021) reported advantages in using ConformalLayers for $k \geq 9$.

Considering the CO_2 emission rate plot in Fig. 4 (bottom), we also have constant behavior for DkNetCL. In contrast, the emission rates of DkNet per batch increase as the neural network becomes deeper. This time, however, the mean

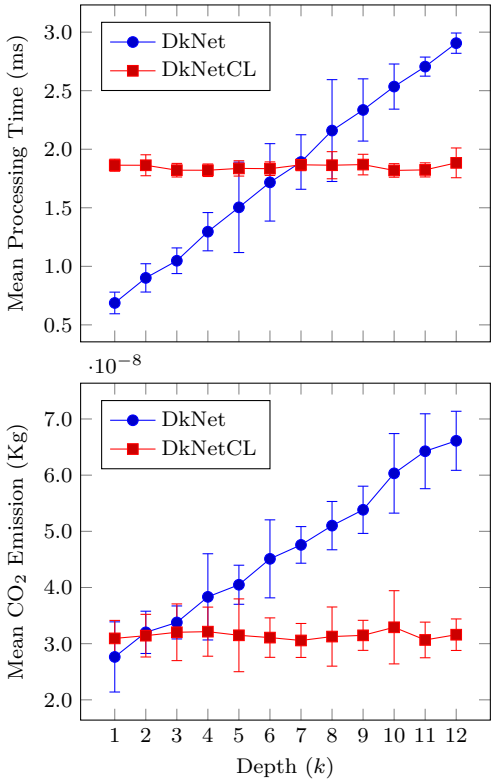

Fig. 4: Mean processing time and CO_2 emission rates for batches of size 64.

emission rate of ConformalLayers-based networks becomes more advantageous for $k \geq 3$. Here, the p-value of 0.44 shows a tie for $k = 2$. For all other cases the p-value is less than 0.05.

The second analysis aims to evaluate and compare the robustness of classification networks in the presence of common corruptions found in natural images. Fig. 5 summarizes the mean top-1 error rates (*i.e.,* $\frac{1}{5}\sum_{s=1}^{5} E_{s,c}^{f}$, in %) computed on the five levels of severity of each type of corruption for two sets of networks: BaseReLUNet vs. BaseReSProNet and LeNet vs. LeNetCL vs. LeNetBinary vs. LeNetMasked. Notice that the standard deviations of quantization-based networks (LeNetBinary and LeNetMasked) are smaller, which in practical terms, means that changing the severity level impacts less on those approaches. Such approaches, however, present the highest error rate in the experiments. The ConformaLayers-based approaches, on the other hand, present a sweet spot between robustness to corruption severity and error rate, which can be seen when comparing BaseReLUNet vs. BaseReSProNet and LeNet vs. LeNetCL. Also, notice that the distance between the red and blue dots are smaller for LeNet-based networks than for single-layer networks, which means that as the network became more elaborate (*i.e.,* from BaseReSProNet to LeNetCL), the difference between the errors became smaller. According to the Welch's t-test, the null hypothesis that two populations have equal mean top-1 error rates is not rejected for corruptions *Contrast* ($p = 0.77$), *Fog* ($p = 0.06$), and *Impulse Noise* ($p = 0.06$) for BaseReLUNet and BaseReSProNet, and *Contrast* ($p = 0.80$), *Fog* ($p = 0.40$), *Gaussian Blur* ($p = 0.20$), *Motion Blur*

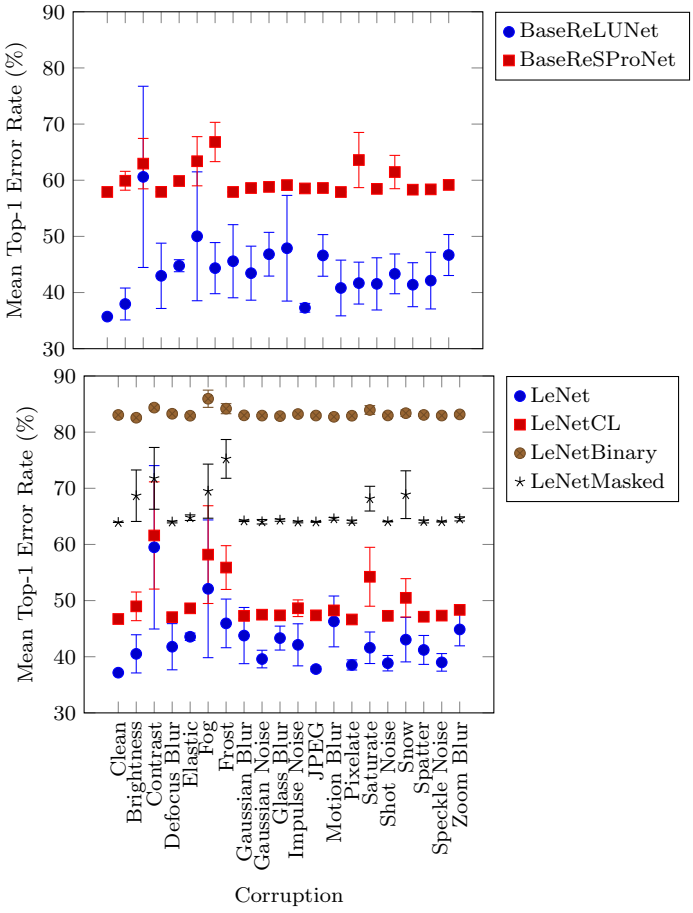

Fig. 5: Mean top-1 error rate (%) with confidence interval of one standard deviation of the networks f applied on corruption c with severity level $1 \leq s \leq 5$.

($p = 0.39$), and *Zoom Blur* ($p = 0.05$) for LeNet and LeNetCL. It means that for one layer and LeNet-based networks, the mean error rates are virtually the same for, respectively, three and five kinds of corruption.

An interesting analysis comes from the fact that, although the ConformalLayers-based approach presents higher error rates than LeNet for corrupted images, they do not change much compared to the clean dataset's error rate. We deep dive into this characteristic by looking at Table 2, which presents the evaluation of the neural networks under the metrics mCE (14) and Relative mCE (16). Our first observation is that neural networks built with a single layer using ConformalLayers tend to misclassify more that the vanilla LeNet model. As can be seen, when using BaseReLUNet as a baseline network and moving from BaseReSProNet to LeNetCL, we notice the reduction of the mCE by 17.12%. When moving from BaseReLUNet to LeNet, under the same baseline, we observe a decrease of 2.59%. Notice that the mCE distance between the vanilla LeNet and the LeNetCL networks (15.33 percentage points) is smaller than the mCE distance of the single-layer networks (36.04 percentage points). Therefore, the increase in complexity of networks built with ConformalLayers suggests more beneficial for corruption tolerance than on networks with non-associative layers. A similar result is observed using LeNet as the baseline network. For the quantization-based networks (LeNetBinary and

Table 2: Mean Corruption Error (mCE) and Relative mCE, in %.

Network	Baseline Network			
	BaseReLUNet		LeNet	
	mCE	Relative mCE	mCE	Relative mCE
BaseReLUNet	100.00	100.00	102.89	186.27
BaseReSProNet	136.04	27.53	139.70	36.68
LeNet	97.41	67.17	100.00	100.00
LeNetCL	112.74	35.77	115.70	46.54
LeNetBinary	189.14	1.03	194.39	0.32
LeNetMasked	149.99	31.37	153.97	32.38

LeNetMasked), one can notice the higher mCE values (89.14 and 49.99 percentage points from the BaseReLUNet as baseline, respectively). These higher mCE values reveal a drawback of these approaches in LeNet-based architectures, while the smaller Relative mCE values show that they suffer less as we add corruption to the data.

When looking at the Relative mCE column of Table 2, one should notice the considerably smaller values for ConformalLayers-based architectures when compared to vanilla LeNet. Since this metric describes how the classifier degrades in the presence of noise, relatively to the error on a clean dataset, we can state that the approaches based on ConformalLayers are more robust to these kinds of corruptions present in CIFAR-10-C than conventional approaches, with relative mCE ranging from 27.53% to 35.77% using BaseReLUNet as the baseline and from 36.68% to 46.54% using LeNet as the baseline. These results show that, despite ConformalLayers presented a higher mean corruption error when compared to the standard non-associative approach, the classifier degrades proportionally less in the presence of corruption than the vanilla non-associative approaches when we compare to the error on the clean dataset.

6. Conclusion and Future Work

This paper extends our previous work in three ways: (i) by presenting an updated analysis of the processing time of ConformalLayers-based CNNs with varying depth; (ii) by discussing their performance regarding CO_2 emission rates; and (iii) by assessing the robustness of single layer and LeNet-based models to common corruptions found in natural images.

We believe that there are several directions to be explored to analyze the impact of ConformalLayers in the design of sequential neural networks as solution to open problems in deep learning. The purpose of this work is to illustrate one of these directions, which is to assess the robustness of ConformalLayers-based networks to corruption in natural images.

In a complementary direction, we show that our approach leads to lower CO_2 emission rates and requires fewer computational resources. We hope this observation motivates the community to investigate further applications of ConformalLayers.

Acknowledgments

This work was partially supported by FAPERJ (E-26/202.718/2018) and CNPq (311.037/2017-8) agencies.

References

Addepalli, S., Jain, S., Sriramanan, G., Khare, S., Radhakrishnan, V.B., 2021. Towards achieving adversarial robustness beyond perceptual limits, in: Proc. ICML Workshop.

Benz, P., Zhang, C., Karjauv, A., Kweon, I.S., 2021. Revisiting batch normalization for improving corruption robustness, in: Proc. WACV, pp. 494–503.

Bergstra, J., Yamins, D., Cox, D., 2013. Making a science of model search: hyperparameter optimization in hundreds of dimensions for vision architectures, in: ICML, pp. 115–123.

Choy, C., Gwak, J., Savarese, S., 2019. 4D Spatio-temporal ConvNets: Minkowski convolutional neural networks, in: Proc. CVPR, pp. 3075–3084.

Dorst, L., Fontijne, D., Mann, S., 2010. Geometric algebra for computer science: an object-oriented approach to geometry. Elsevier.

Gonzalez, R.C., Woods, R.E., 2008. Digital image processing. 3 ed., Pearson.

Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep learning. MIT.

Hendrycks, D., Dietterich, T., 2019. Benchmarking neural network robustness to common corruptions and perturbations. Proc. ICLR .

Hendrycks, D., Mu, N., Cubuk, E.D., Zoph, B., Gilmer, J., Lakshminarayanan, B., 2020. AugMix: a simple data processing method to improve robustness and uncertainty. Proc. ICLR .

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y., 2016. Binarized neural networks, in: NeurIPS, pp. 4114–4122.

Kireev, K., Andriushchenko, M., Flammarion, N., 2021. On the effectiveness of adversarial training against common corruptions. arXiv:2103.02325.

Krizhevsky, A., 2009. Learning multiple layers of features from tiny images. Technical Report. University of Toronto.

Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012. ImageNet classification with deep convolutional neural networks, in: Proc. NIPS.

Lecun, Y., Bottou, L., Bengio, Y., Haffner, P., 1998. Gradient-based learning applied to document recognition. Proc. IEEE 86, 2278–2324.

Lee, J., Won, T., Hong, K., 2020. Compounding the performance improvements of assembled techniques in a convolutional neural network. arXiv:2001.06268.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., Talwalkar, A., 2017. Hyperband: a novel bandit-based approach to hyperparameter optimization. J. Mach. Learn. Res. 18, 6765–6816.

Lopes, R.G., Yin, D., Poole, B., Gilmer, J., Cubuk, E.D., 2020. Improving robustness without sacrificing accuracy with patch Gaussian augmentation. arXiv:1906.02611.

Lyzhov, A., Molchanova, Y., Ashukha, A., Molchanov, D., Vetrov, D., 2020. Greedy policy search: a simple baseline for learnable test-time augmentation, in: Proc. UAI, pp. 1308–1317.

Milyaev, S., Laptev, I., 2017. Towards reliable object detection in noisy images. Pattern Recognit. Image Anal. 27, 713–722.

Reina, P., Menéndez, A., Menéndez, J., Bressan, G., Ruggeiro, W., 2021. Understanding the impact of image quality in face processing algorithms, in: Proc. IMPROVE, pp. 145–152.

Rusak, E., Schott, L., Zimmermann, R.S., Bitterwolf, J., Bringmann, O., Bethge, M., Brendel, W., 2020. A simple way to make neural networks robust against diverse image corruptions, in: Proc. ECCV, pp. 53–69.

Schmidt, V., Goyal, K., Joshi, A., Feld, B., Conell, L., Laskaris, N., Blank, D., Wilson, J., Friedler, S., Luccioni, S., 2021. CodeCarbon: estimate and track carbon emissions from machine learning computing. `https://github.com/mlco2/codecarbon`.

Sousa, E.V., Fernandes, L.A.F., Vasconcelos, C.N., 2021. ConformalLayers: a non-linear sequential neural network with associative layers, in: Proc. SIBGRAPI, pp. 386–393.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I.J., Fergus, R., 2014. Intriguing properties of neural networks, in: Proc. ICLR.

Welch, B.L., 1947. The generalization of Student's problem when several different population variances are involved. Biometrika 34, 28–35.

Yin, D., Lopes, R.G., Shlens, J., Cubuk, E.D., Gilmer, J., 2019. A Fourier perspective on model robustness in computer vision, in: Proc. NeurIPS, Curran Associates, Inc.. pp. 13276–13286.

Young, W.H., 1912. On the multiplication of successions of Fourier constants. Proc. R. Soc. Lond. Series A, Containing Papers of a Mathematical and Physical Character 87, 331–339.

Zhou, H., Lan, J., Liu, R., Yosinski, J., 2019. Deconstructing lottery tickets: zeros, signs, and the supermask, in: NeurIPS, pp. 3592–3602.