Enhanced robustness of convolutional networks with a push–pull inhibition layer

Strisciuglio, Nicola; Lopez-Antequera, Manuel; Petkov, Nicolai

Published in:
Neural Computing and Applications

DOI:
10.1007/s00521-020-04751-8

Publication date:
2020

Neural Computing and Applications (2020) 32:17957–17971
https://doi.org/10.1007/s00521-020-04751-8

S.I. : DEVELOPING NATURE-INSPIRED INTELLIGENCE BY NEURAL SYSTEMS

Enhanced robustness of convolutional networks with a push–pull inhibition layer

Nicola Strisciuglio[1,2] · Manuel Lopez-Antequera[1,3] · Nicolai Petkov[1]

Received: 25 April 2019 / Accepted: 23 January 2020 / Published online: 5 February 2020

Abstract
Convolutional neural networks (CNNs) lack robustness to test image corruptions that are not seen during training. In this paper, we propose a new layer for CNNs that increases their robustness to several types of corruptions of the input images. We call it a 'push–pull' layer and compute its response as the combination of two half-wave rectified convolutions, with kernels of different size and opposite polarity. Its implementation is based on a biologically motivated model of certain neurons in the visual system that exhibit response suppression, known as push–pull inhibition. We validate our method by replacing the first convolutional layer of the LeNet, ResNet and DenseNet architectures with our push–pull layer. We train the networks on original training images from the MNIST and CIFAR data sets and test them on images with several corruptions, of different types and severities, that are unseen by the training process. We experiment with various configurations of the ResNet and DenseNet models on a benchmark test set with typical image corruptions constructed on the CIFAR test images. We demonstrate that our push–pull layer contributes to a considerable improvement in robustness of classification of corrupted images, while maintaining state-of-the-art performance on the original image classification task. We released the code and trained models at the url http://github.com/nicstrisc/Push-Pull-CNN-layer.

Keywords Convolutional neural networks · Image corruption · Network robustness · Neuron response inhibition · push–pull layer

1 Introduction

Convolutional neural networks (CNNs) are routinely used in many problems of image processing and computer vision, such as large-scale image classification [22], semantic segmentation [6], optical flow [20], stereo matching [36], among others. They became a de facto standard in computer vision and are gaining increasing research interest. The success of CNNs is attributable to their ability of learning representations of input training data in a hierarchical way, which yields state-of-the-art results in a wide range of tasks. The availability of appropriate hardware, namely GPUs and deep learning dedicated architectures, to facilitate huge amounts of required computations has favored their spread, use and improvement.

A number of breakthroughs in image classification were achieved by end-to-end training of deeper and deeper architectures. AlexNet [22], VGGNet [35] and GoogleNet [41], which were composed of eight, 19 and 22 layers, respectively, pushed forward the state-of-the-art results on large-scale image classification. Subsequently, learning of extremely deep networks was made possible with ResNet [16], whose architecture based on stacked bottleneck layers and residual blocks helped alleviate the problem of vanishing gradients. Such very deep networks, with hundreds or even a thousand layers, contributed to push the classification accuracy even higher on many benchmark data sets for image classification and object detection. With

✉ Nicola Strisciuglio
n.strisciuglio@utwente.nl

[1] Bernoulli Institute for Mathematics, Computer Science and Artificial Intelligence, University of Groningen, Groningen, The Netherlands

[2] Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, Enschede, The Netherlands

[3] MAPIR Group, Biomedical Research Institute of Málaga (IBIMA), University of Málaga, Málaga, Spain

WideResNet [48], it was shown that shallower but wider networks can achieve better classification results without increasing the number of learned parameters. In [18], a densely connected convolutional network named DenseNet was proposed that deploy forward connection of the response maps at a given layer to all subsequent layers. This mechanism allowed to reduce the total number of parameters to be learned, while achieving state-of-the-art results on ImageNet classification.

These networks suffer from reliability problems due to instability [49], i.e., small changes of the input cause big changes of the output. In [17], the authors demonstrated that the enormous increase in accuracy achieved by recently published networks on benchmark classification tasks (e.g., ImageNet and CIFAR) does not couple with an improvement of robustness to classification of corrupted test samples. They showed that, by corrupting the test images with noise, blur, fog and other common transformations, the performance of SOTA networks drops considerably similar to early CNN methods, namely AlexNet. Some approaches to increase the stability of deep neural networks to noisy images make use of data augmentation, i.e., new training images are created by adding noise to the original ones. This approach, however, improves robustness only to those classes of perturbation of the images represented by the augmented training data and requires that this robustness is learned: it is not intrinsic to the network architecture. In [49], a structured solution to the problem was proposed, where a loss function that controls the optimization of robustness against noisy images was introduced.

In this paper, instead, we use prior knowledge about the visual system of the brain to guide the design of a new component for CNN architectures: we propose a new layer called *push–pull layer*. We were inspired by the push–pull inhibition phenomenon that is exhibited by some neurons in area V1 of the visual system of the brain [42]. Such neurons are tuned to detect specific visual stimuli, but respond to such stimuli also when they are heavily corrupted by noise. The inclusion of this layer in the network architecture contributes to an increase in robustness of the model to various corruptions of the input images, *while maintaining state-of-the-art performance on the original image classification task*. This comes without an increase in the number of parameters and with a negligible increase in computation. Furthermore, the proposed push–pull layer can be used in any CNN architecture, being a general approach to enhance network robustness. Our contributions are summarized as follows:

- We propose a new type of layer for CNN architectures.
- We validate our method by including the proposed push–pull layer into state-of-the-art residual and dense network architectures, namely ResNet and DenseNet, and training them on the task of image classification. We study the effect of using the proposed push–pull layer in the first layer of CNN architectures. It intrinsically embues the network with enhanced robustness to image corruptions without increasing the model size.
- We show the impact of the proposed method by comparing the performance of state-of-the-art networks with and without the push–pull layer on the classification of corrupted images. We experimented on a benchmark version of the CIFAR data set, which contains different types and severities of corruptions [17]. Our proposal improves classification accuracy of corrupted images while maintaining performance on the original images.
- We provide an implementation of the proposed push–pull layer as a new layer for CNNs in PyTorch, available at the url http://github.com/nicstrisc/Push-Pull-CNN-layer.

2 Related works

Data augmentation and robustness to image corruptions. The success of CNNs and deep learning in general can be attributed to the representation capacity of these models, enabled by their size and hierarchical nature. However, this large capacity can become problematic as it can be hard to avoid overfitting to the training set. Early work achieving success on large-scale image classification [22] noted this and included data augmentation schemes, where training samples were modified by means of transformations of the input image that do not modify the label, such as rotations, scaling, cropping, and so on [22]. Data augmentation can also be used to allow the network to learn invariances to other transformations not present on the training set but that can be expected to appear when deploying the network.

The main drawback of data augmentation is that the networks acquire robustness only to the classes of perturbations used for training [49]. Many studies have been published that tackle the problem of making the networks robust to various kinds of input distortions. In [45], blurred images were used to fine-tune networks and it was demonstrated that one type of blurring does not generalize to others. Heavy data augmentation to increase network robustness can thus cause underfitting, as confirmed in [13, 49]. Furthermore, human performance was demonstrated to be superior even to fine-tuned networks on Gaussian noise and blur in [10].

Although data augmentation contributes, to some extent, to increase the generalization capabilities of classification models with respect to certain data transformations, real-world networks need to incorporate mechanisms that intrinsically increase their robustness to corruption of input data.

Recently, several data sets that contain test images with various corruptions and perturbations have been released, with the aim of benchmarking network robustness to input distortions. In [43, 44] data sets with corrupted traffic sign images and objects were proposed. Recently, a large benchmark data set constructed by adding 15 different types of corruption, each with five levels of severity, and 10 perturbations to the test images of the ImageNet and CIFAR data sets was released [17].

Adversarial attacks. An adversarial attack consists of sightly distorting an input sample for the purpose of confusing a classifier [40]. Recently, algorithms have been developed that produce the smallest possible distortions of input samples that fool a classification model [1]. In the case of images, adversarial attacks create additive signals in the RGB space that make changes imperceptible to the human eye which drift their representations through the decision boundaries of another class. In this light, adversarial attacks can be considered the worst case of input corruption that networks can be subjected to.

On the one hand, in recent years various adversarial attacks have been developed, such as Fast Gradient Sign Method (FGSM) [14], iterative methods [23, 29] and DeepFool [32], Carlini and Wagner (C&W) [8], Universal Adversarial Perturbations [33], and black-box attacks UPSET and ANGRI [1]. On the other hand, defensive algorithms were also developed for specific adversarial attacks [28, 31, 34].

Although adversarial attacks and defenses are important topics to machine learning research, in this work we study a different kind of model robustness. We focus on robustness against image corruptions determined by noise, blur, elastic transformations, fog and so on, which are typical of many computer vision tasks, instead of adversarial attacks that alter the test samples with unnoticeable modifications [17].

Prior knowledge in deep architectures. Domain specific knowledge can be used to guide the design of deep neural network architectures. In this way, they better represent the problem to be learned in order to increase efficiency or performance. For example, convolutional neural networks are a subset of general neural networks that encode translational invariance on the image plane.

Specific architectures or modules have been designed to encode properties of other problems. For instance, steerable CNNs include layers of steerable filters to compute orientation-equivariant feature response maps [46]. They achieve rotational equivariance by computing the responses of feature maps at a given set of orientations. In Harmonic CNNs, rotation equivariance was achieved by substituting convolutional filters with circular harmonics [47]. In [9], a formulation of spherical cross-correlation was proposed, enabling the design of Spherical CNNs, suitable for application on spherical images.

Biologically inspired models. One of the first biologically inspired models for Computer Vision was the neocognitron network [12]. The architecture consisted of layers of S-cells and C-cells, which were models of simple and complex cells in the visual system of the brain. The network was trained *without a teacher*, in a self-organizing fashion. As a result of the training process, the neocognitron network had a structure similar to the hierarchical model of the visual system formalized by Hubel and Wiesel [19].

The filters learned in the first layer of a CNN trained on natural images resemble Gabor kernels and the receptive fields of neurons in area V1 of the visual system of the brain [30]. This strengthens the connection between CNN models and the visual system. However, the convolutions used in CNNs are linear operations and are not able to correctly model some nonlinear properties of neurons in the visual system, e.g., cross orientation suppression and response saturation. These properties were achieved by a nonlinear model of simple cells in area V1, named CORF (Combination of Receptive Fields), used in image processing for contour detection [3] or for delineation of elongated structures [5, 37]. Neuro-physiological models of inhibition phenomena in the human visual system have also been included in image processing tools [4, 38].

A layer of nonlinear convolutions inside CNNs was proposed in [50]. The authors were inspired by studies of nonlinear processes in early stages of the visual system, and modeled them by means of Volterra convolutions.

3 Method: CNN augmentation with a push–pull layer

We propose a new type layer that can be used in existing CNN architectures to improve their robustness to different t of image corruptions. We call it *push–pull* layer as its design is inspired by the functions of some neurons in area V1 of the visual system of the brain that exhibit a phenomenon known as *push–pull* inhibition [21]. Such neurons have excitatory and inhibitory receptive fields that respond to stimuli of opposite polarity. Their responses are combined in such a way that these neurons strongly

respond to specific visual stimuli, also when they are corrupted by noise. We provide a wider discussion about the biological inspiration of the proposed push–pull layer in "Appendix 1". In the rest of the section, we explain the details of the proposed layer.

3.1 Implementation

We design the push–pull layer $\mathcal{P}(I)$ using two convolutional kernels, which we call *push* and *pull* kernels. They model the *excitatory* and the *inhibitory* receptive fields of a push–pull neuron, respectively. The pull kernel typically has a larger support region than that of the push kernel and its weights are computed by inverting and upsampling the push kernel [42]. We implement *push–pull* inhibition by subtracting a fraction α of the response of the pull component from that of the push component. We model the activation functions of the push and pull receptive fields by using nonlinearities after the computation of the push and pull response maps. In Fig. 1 we show an architectural sketch of the proposed layer.

We define the response of a push–pull layer as:

$$\mathcal{P}(I) = \Theta(k * I) - \alpha\Theta(-k_{\uparrow h} * I)$$

where $\Theta(\cdot)$ is a rectifier linear unit (ReLU) function, α is a weighting factor for the response of the pull component which we call inhibition strength. Finally, $\uparrow h$ indicates upsampling of the push kernel $k(\cdot)$ by a scale factor $h > 1$.

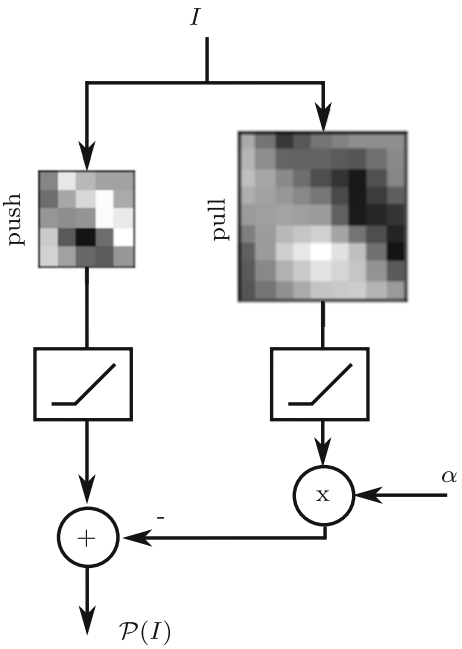

Fig. 1 Architectural scheme of the push–pull layer. The input array I is convolved with two kernels, namely the push and pull kernels. The resulting response maps are rectified and subsequently combined by weighted sum. The pull kernel is an upsampled and inverted version of the push kernel

Let s be the width (or height) of the push kernel, the width (or height) $\hat{s}$ of pull kernel is computed as $\hat{s} = \lfloor s \cdot h \rfloor + 1 - (\lfloor s \cdot h \rfloor \bmod 2)$. h and α are hyper-parameters of the proposed push–pull layer and their value has to be set by the user. We discuss their configuration in Sect. 4. We use ReLU functions to implement the nonlinear behavior of the push–pull neurons as they provide an effective and simple way to model the activation function of convolution filters. However, one may use other nonlinear functions, e.g., the hyperbolic tangent function.

During training, only the weights of the push kernel are updated. The weights of the pull kernel are derived from those of the actual push kernel as explained above, i.e., at each forward step the pull kernel is generated online via differentiable upsampling and inverting operations. The implementation of the push–pull layer ensures that the gradient flows back toward both the push and pull kernel and that the weights of the push kernel are updated accordingly. In this way, the effect of the pull component is taken into account when the gradient is back-propagated through the pull kernel.

In the first row of Fig. 2, we show an image from the MNIST data set corrupted by Gaussian noise of increasing severity. We also display the response map of a convolutional kernel only (second row) in comparison with that of a push–pull layer (third row). One can observe how the push–pull layer is able to detect the feature of interest, which was learned in the training phase, more reliably than the convolution (push only) kernel, even when the input is corrupted by noise of high severity. The enhanced robustness to image corruption is due to the effect of the pull component, which suppresses the responses of the push kernel caused by noisy and spurious patterns.

3.2 Use of the push–pull layer

We implemented a push–pull layer for CNNs in PyTorch and deploy it by substituting the first convolutional layer of existing CNN architectures. In Fig. 3, we show sketches of modified LeNet, ResNet and DenseNet architectures. We replaced the first convolutional layer *conv1* with our *push–pull* layer. The resulting architecture is surrounded by the dashed contour line. Hereinafter, we use the suffix '-PP' to indicate that the concerned network deploys a push–pull layer as the first layer, instead of a convolutional layer.

In this work, we train the modified architectures from scratch. One can also replace the first layer of convolutions of an already trained model with our push–pull layer. In such case, however, the model requires a fine-tuning procedure so that the layers succeeding the push–pull layer can adapt to the new response maps, as the responses of the push–pull layer are different from those of the convolutional layers (see the second and third row in Fig. 2).

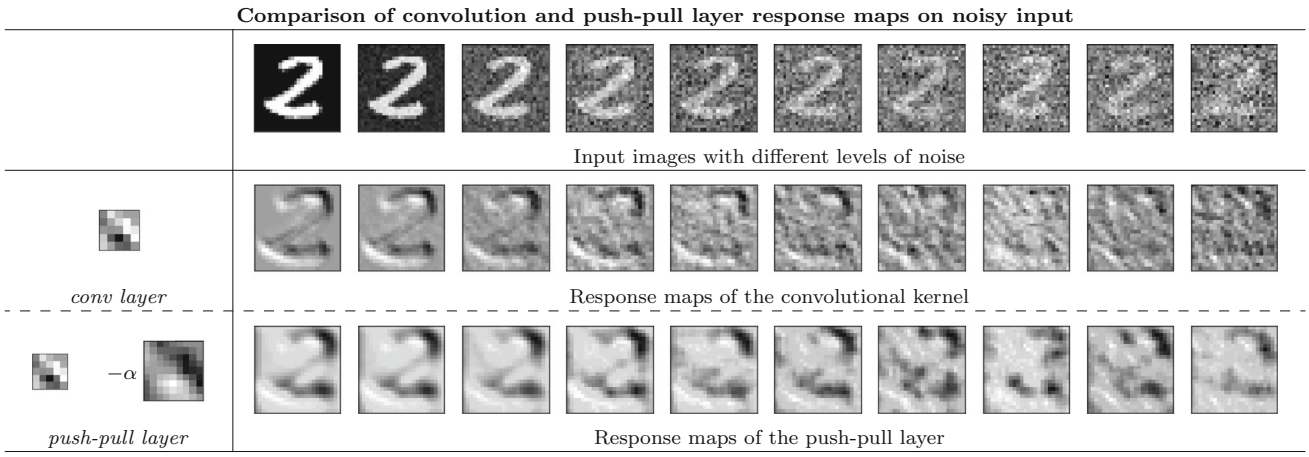

Fig. 2 Images of a digit from the MNIST data set perturbed by added Gaussian noise of increasing severity (first row). The response maps of a convolutional kernel in the second row show instability with respect to perturbed inputs. Our push–pull layer is more robust to noise as shown in the response maps in the third row

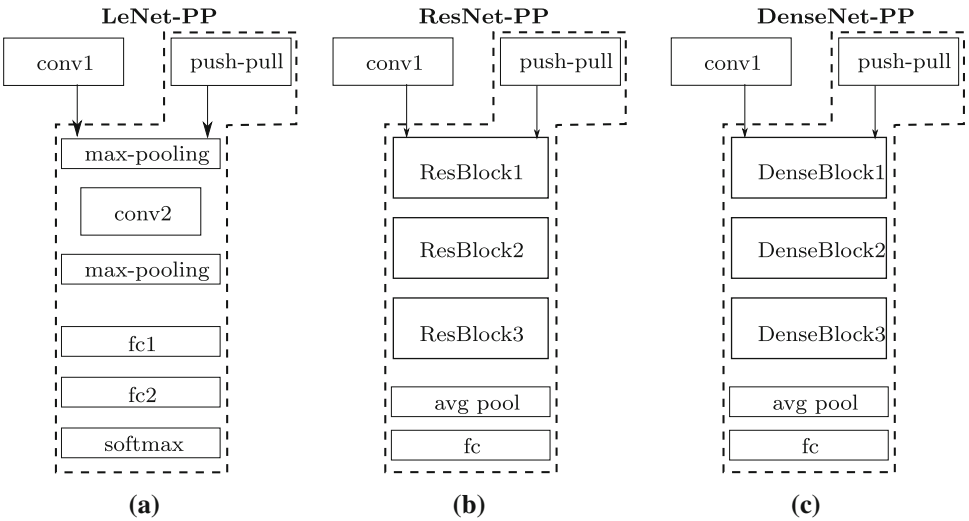

Fig. 3 Modified **a** LeNet, **b** ResNet and **c** DenseNet architectures. We substitute the first layer of convolutions (conv1) with our push–pull layer. The suffix 'PP' in the network names stands for push–pull. The new modified networks are highlighted by dashed lines

In principle, a push–pull layer can be used at any depth level in deep network architectures as a substitute of any convolutional layer. However, its implementation is related to the behavior and functions of some neurons in early stages of the visual system of the brain, where low-level processing of the visual stimuli is performed. In this work, we thus focus on analyzing the effect of using the proposed push–pull layer only as first layer of the networks and to evaluate its contribution to enhance the robustness of the networks to corruptions of the input test images.

4 Experiments and results

We carried out extensive experiments to validate the effectiveness of our push–pull layer for improving the robustness of existing networks to perturbations of the input image. We include the push–pull layer in the LeNet, ResNet and DenseNet architectures, by replacing the first convolutional layer.

We train the LeNet, ResNet and DenseNet networks on non-corrupted training images from the MNIST and CIFAR data sets, and test on images with several corruptions, of different types and severities, that are unseen by the training process. We compare the results obtained by CNNs that employ a push–pull layer as substitute of the first convolutional layer with those from a standard CNN. The results that we report were obtained by replacing the first convolutional layer with a push–pull layer with upsampling factor $h = 2$ and inhibition strength $\alpha = 1$. In Sect. 4.3, we study the sensitivity of the classification performance with respect to different configurations of the push–pull layer. h and α are hyper-parameters of the push–pull layer and their value trades off the accuracy of the model on clean data and its robustness to corrupted data. For these experiments, we chose the configuration

according to experimental results and previous work in including the push–pull inhibition model into an operator for line detection in noisy images, called RUSTICO [38, 39].

4.1 LeNet on MNIST

The MNIST data set is composed of 60 k images of handwritten digits (of size 28×28 pixels), divided into 50 k training and 10 k test images. The data set has been widely used in computer vision to benchmark algorithms for object classification. LeNet is one of the first convolutional networks [24], and achieved remarkable results on the MNIST data set. It is considered one of the milestones of the development of CNNs. We use it in the experiments for the simplicity of its architecture, which allows to better understand the effect of the push–pull layer on the robustness of the network to input image corruptions.

4.1.1 Training

The original LeNet model is composed of two convolutional layers, the first with six filters and the second with 16, after each a max-pooling layer is placed. Three fully connected layers are added on top of the convolutional layers to perform classification. The last layer is composed as many neurons as the number of classes (10 in our case). We configured different LeNet models by changing the number of convolutional filters in the first and second layer (note that the size of the fully connected layers changes accordingly to the number of filters in the second convolutional layer). We implemented push–pull versions of LeNet by substituting the first convolutional layer with our push–pull layer. In Table 1, we report details on the configuration of the LeNet models modified with the push–pull layers. The letter 'P' in the model names indicate the use of the proposed push–pull layer.

We trained all LeNet models using stochastic gradient descent (SGD) on the original training set of the MNIST data set for 90 epochs. We set an initial learning rate of 0.1 and decrease it by a factor of 10 at epochs 30 and 60. We configure the SGD algorithm with Nesterov momentum equal to 0.9 and weight decay of $5 \cdot 10^{-4}$.

4.1.2 Results

We report the results achieved on the MNIST test set perturbed with Gaussian noise of increasing variance in Fig. 4. When the variance of the noise increases above $\sigma^2 = 0.1$, the improvement of performance determined by the use of the push–pull layer is noticeable ($A = 86.5\%$, PA = 87.1%—$B = 73.91\%$, PB = 87.2%—$C = 86.2\%$, PC = 85.14%—$D = 78.62\%$, PD = 82%, for Gaussian noise with $\sigma^2 = 0.2$), revealing an increase in the generalization capabilities of the networks and of their robustness to noise.

Generally the use of the push–pull layer contributes to increase the representation capacity of the network and its generalization capabilities with respect to unknown corruptions of the input data. However, in the case of the model C, the use of the push–pull layer worsens the classification results on data corrupted with Gaussian noise. Our interpretation is that the small number of features computed at the first layer do not provide the following layers (which remain unmodified having a large number of channels/features relative to the first layer) a representation with enough capacity to achieve satisfactory generalization capabilities. This effect is mitigated in model D, where a smaller network is configured after the first layers. The lower capacity of the sub-network that takes as input the response of the push–pull layer thus reduces the chances to overfit to the training data. The largest improvement is

Table 1 Configurations of the LeNet architecture used in the experiments on the MNIST data set

Model	ConvNet		FCNet
	1st layer	2nd layer	
A	6 (c)	16 (c)	128, 64, 10
B	6 (c)	8 (c)	64, 32, 10
C	4 (c)	16 (c)	128, 64, 10
D	4 (c)	8 (c)	64, 32, 10
PA	6 (pp)	16 (c)	128, 64, 10
PB	6 (pp)	8 (c)	64, 32, 10
PC	4 (pp)	16 (c)	128, 64, 10
PD	4 (pp)	8 (c)	64, 32, 10

The label (c) indicates a convolutional layer, while (pp) a push–pull layer

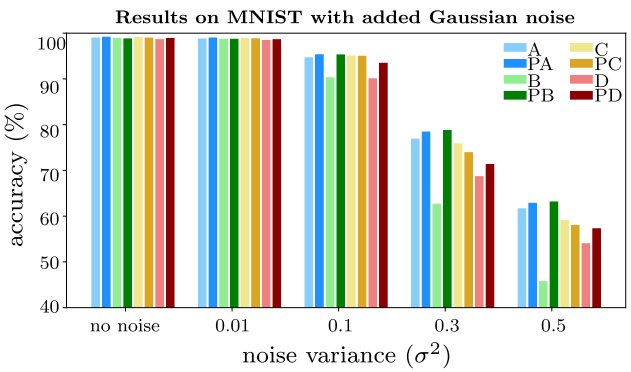

Fig. 4 Results of LeNet (lighter colors—A, B, C, D) and LeNet-PP (darker colors—PA, PB, PC, PD) on the MNIST test set images corrupted by added Gaussian noise of increasing severity (color figure online)

obtained by the model PB with respect to its convolutional counterpart B. As shown in Fig. 2, the push–pull layer computes more stable response maps than those of the convolutional layers in presence of image corruptions. We conjecture that model PA and PC are more subject to specialization due to the larger size of the following layers, model PB achieves the best generalization performance. This results from the combination of the push–pull layer output with a network of smaller size, whose optimization is simpler and can more easily reach a better local minimum of the loss function.

In Fig. 5, we compare the results achieved by the different LeNet models with the push–pull layer (darker colors—PA, PB, PC, PD) with those of the original LeNet (lighter colors—A, B, C, D) on the MNIST test set images perturbed by change of contrast and addition of Poisson noise. We use different factors C to increase or decrease the contrast of the input image I, and produce new images $I_C = (I - 0.5) * C + 0.5$.

The LeNet-PP models considerably outperform their convolution-only counterparts when the contrast of noisy test images decreases and the images are corrupted by Poisson noise. It is interesting that models A and D show a considerable drop of classification performance when the contrast level is lower than $C = 0.5$. We hypothesize that this is probably due to specialization of the networks on the characteristics of the images in the training set. Model B achieves more stable results when the contrast level is higher or equal to 0.3. Similarly to the case of images corrupted with Gaussian noise, model PB achieves a better stability and robustness to corruption of the input images. Models PA and PD largely benefit from the presence of the push–pull layer, that is mostly due to the computation of response maps that are more robust to input corruptions

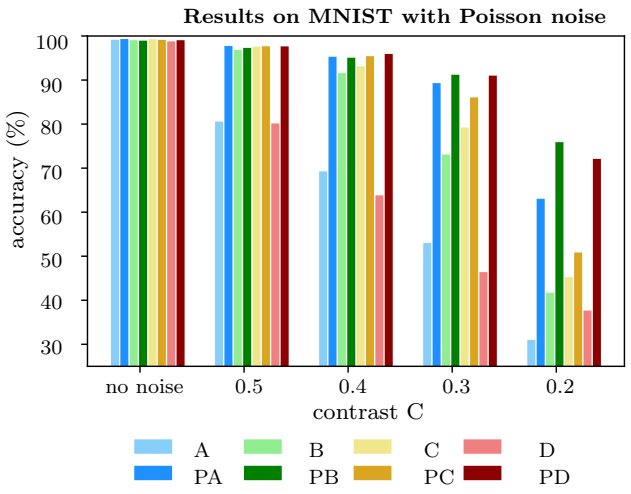

Fig. 5 Results achieved by LeNet (lighter color bars) and LeNet-PP (darker color bars) on the MNIST test set images corrupted with changes of contrast and Poisson noise (color figure online)

and favor a more stable further processing in the network. We observed that using the push–pull layer allows smaller networks to achieve more robustness than larger networks without push–pull layers (see B and D, which are roughly half-size w.r.t. A and C).

It is worth pointing out that in all cases, the classification accuracy on the original test set (without corruption) is not substantially affected by the use of the push–pull layer ($A = 98.93\%$, PA $= 99.1\%$—$B = 98.85\%$, PB $= 98.78\%$—$C = 99.06\%$, PC $= 98.91\%$—$D = 98.58\%$, PD $= 98.84\%$).

4.2 ResNet and DenseNet on CIFAR

4.2.1 CIFAR corruption benchmark data set

The CIFAR-10 is a data set for benchmarking algorithms for image and object recognition. It is composed of 60 k natural images (of size 32×32 pixels) organized in 10 classes and divided into 50 k images for training and 10 k for test.

In this work, we carried out experiments using a modified version of the CIFAR-10 data set, namely the CIFAR-C, where C stands for 'corruption'. It is a benchmark data set constructed by applying common corruptions to the images in the CIFAR test set [17]. The authors released a data set composed of several test sets, each of them corresponding to a particular type of image corruption. The first version of the data set contained 15 corrupted sets, while in the extended version four further corruption types were included. We performed experiments on the complete set of 19 corrupted test sets. Each corruption is applied to the images of the CIFAR-10 data set with five levels of severity, resulting in a total of 90 different versions of the CIFAR-10 test set. The considered corruptions are of four types: *noise* (Gaussian noise, shot noise, impulse noise, speckle noise), *blur* (defocus blur, glass blur, motion blur, Gaussian blur), *weather* (snow, frost, fog, brightness, spatter) and *digital* (contrast, elastic transformation, pixelate, jpeg compression, saturate). In Fig. 6, we show example images from the corrupted test sets of the CIFAR-C data set, with corruption severity $s = 4$.

The main strength point of the CIFAR-C data set is that it contains common image corruptions that occur when applying computer vision algorithms to real-world data and problems. It thus serves as a thorough benchmark case to evaluate the robustness of state-of-the-art CNN algorithms for image classification in real-world conditions, and their generalization capabilities.

4.2.2 Experiments and evaluation

We trained several configurations of ResNet and DenseNet using the training images of the original CIFAR-10 data

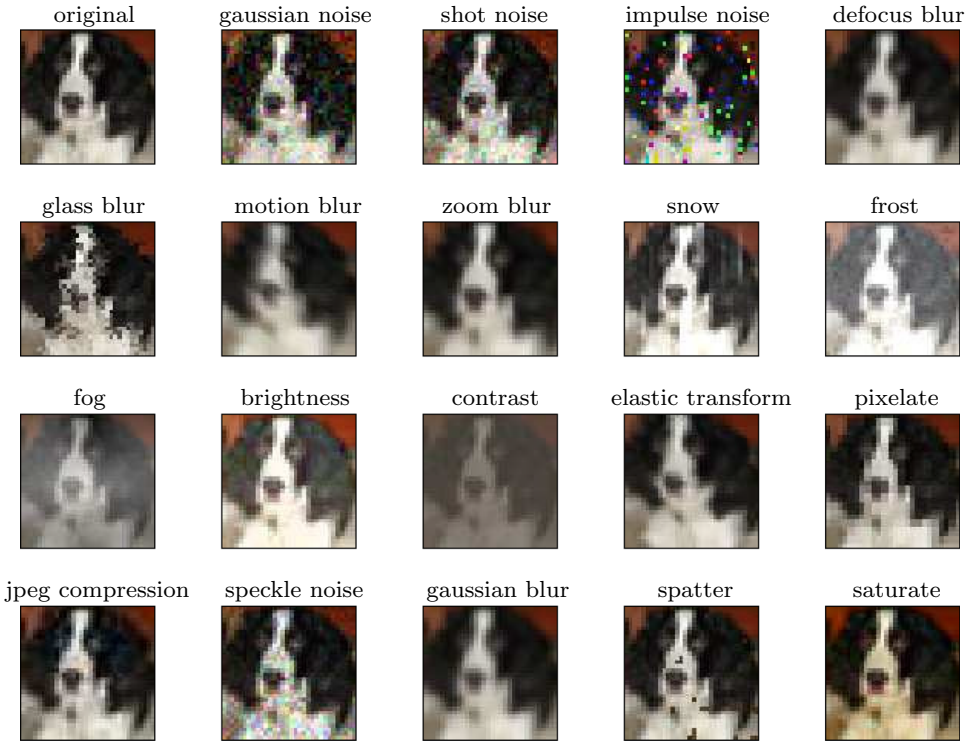

Fig. 6 An image (top-left corner) of the class dog from the original CIFAR-10 test set and examples of corrupted versions of it from the CIFAR-C data set. For all the corrupted images, the corruption severity is $s = 4$

set, on which we apply only the standard data augmentation techniques (i.e., random crop and horizontal flip) introduced in [25]. We subsequently tested the models on the CIFAR-C corrupted test sets, which contain image corruptions that are not present in the training data and not used for data augmentation.

We refer to a ResNet architecture with l layers as ResNet-l [16] and to a DenseNet with l layers and growing factor k as DenseNet-l-k [18]. We evaluated the contribution of the push–pull layer to the performance of models with different depth and, in the case of DenseNet, also with different growing factors. For each network configuration, we train its original version with only convolutional layers and one version with the proposed push–pull layer as substitute of the first convolutional layer, which we refer at with the '-PP' suffix in the model name.

In the original paper, ResNet models are trained for 160 epochs on the CIFAR training set, with a batch size of 128 and an initial learning rate equal to 0.1. The learning rate is reduced by a factor of 10 at epochs 80 and 120. In this work, we trained the ResNet models for 40 more epochs, for a total of 200 epochs, and further reduced the learning rate by a factor of 10 at epoch 160. The extended training was required due to the slightly increased complexity of the learning process determined by the presence of the push–pull layer. In order to guarantee a fair comparison, we trained both models with and without the push–pull layer for 200 epochs. We trained the DenseNet models for 350 epochs, with a batch size equal to 64 and an initial learning rate equal to 0.1. The learning rate is reduced by a factor of 10 at epochs 150, 225 and 300. For both ResNet and DenseNet architectures, we use parameter optimization by means of stochastic gradient descent (SGD) with Nesterov momentum equal to 0.9 and weight decay equal to 10^{-4}.

For evaluation of performance, we computed the Classification Error (E), which is a widely used metric for evaluation of image classification algorithms, and the corruption error (CE). The CE was introduced in [17] and is a weighted average error across the different types and severities of corruption applied to the CIFAR test set. Given a classifier M, the corruption error computed on the images with a corruption c and severity s (with $1 \leq s \leq 5$) is indicated by $\mathrm{CE}^{M}_{c,s}$. Different corruptions cause different levels of difficulty to the classifier. Hence, corruption-specific errors are weighted with the corresponding error obtained by AlexNet, which is taken as the baseline for the evaluation. We report details about the configured AlexNet model and the baseline errors used for normalization of the CE in "Appendix 2". The normalized corruption error on a specific corruption c is defined as:

$$\mathrm{CE}^{M}_{c} = \frac{\sum_{s=1}^{5} \mathrm{CE}^{M}_{c,s}}{\sum_{s=1}^{5} \mathrm{CE}^{\mathrm{AlexNet}}_{c,s}} \tag{1}$$

We summarize the performance of a model by computing the mean corruption error (mCE) across the CE obtained for the different corruption types. The errors achieved by a model M for each corruption type and for each severity are

normalized by the corresponding ones achieved by AlexNet. Subsequently, they are averaged to compute the mCE, which is a percentage measure of the corruption error compared to that achieved by the baseline network.

On the one hand, it can be the case that a classifier is robust to corruptions as the gap between its classification errors on clean and corrupted data is very small. However, such classifier might achieve a high mCE value. On the other hand, a classifier might achieve a very low classification error E on clean data while obtaining high corruption error CE. In [17], the relative corruption error rCE_c^M was introduced, which is a normalized measure of the difference between the classification performance of a model M on clean data and corrupted data. It is defined, for a particular type of corruption c, as:

$$\mathrm{rCE}_c^M = \frac{\sum_{s=1}^{5}\left(\mathrm{CE}_{c,s}^M - \mathrm{CE}_{\mathrm{clean}}^M\right)}{\sum_{s=1}^{5}\left(\mathrm{CE}_{c,s}^{\mathrm{AlexNet}} - \mathrm{CE}_{\mathrm{clean}}^{\mathrm{AlexNet}}\right)} \tag{2}$$

Similarly to the mCE, also the rCE_c^M is normalized by the relative corruption errors of the baseline network AlexNet so as to fairly count the errors achieved on different corruption types. Averaging the rCE_c^M values obtained for each corruption type, we compute the relative mean corruption error (rCE). It measures the average gap between the performance of a network on clean and corrupted data. The lower this measure, the more robust the classifier is to corruptions of the input data.

4.2.3 Results

We evaluated the performance of several ResNet and DenseNet models with (PP) and without (no PP) the proposed push–pull layer as first layer of the architecture. In Fig. 7, we show the average classification error achieved by each considered model with (bars of darker color) and without (bars of lighter color) a push–pull layer as the first layer on the CIFAR-C data set. For all the models, the version with the push–pull layer outperforms (lower classification error E) original convolutional counterpart.

In Table 2, we report the detailed classification errors (E) for each image corruption type that we achieved on the CIFAR-C data set. The push–pull layer contributes to a considerable improvement of the robustness of classification of corrupted images, especially when the images are altered by different types of noise (i.e., Gaussian, shot, impulse and speckle noise). In several cases, the presence of the push–pull layer determines a reduction of the classification error of about 25% (e.g., for ResNet-32 and ResNet-56). Also for image blurs (i.e., defocus, glass, motion, zoom and Gaussian blurring), the model with the

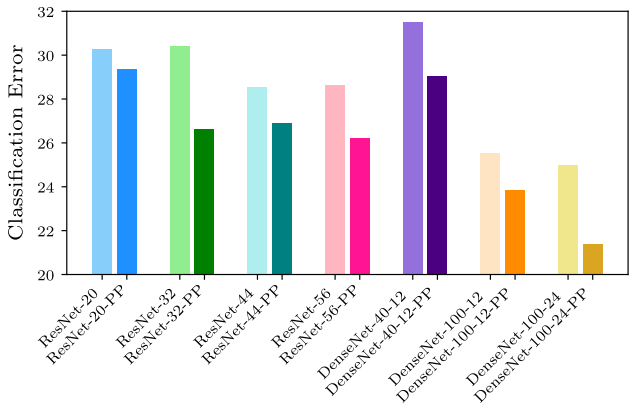

Fig. 7 Comparison of the average classification error achieved by the considered networks on the CIFAR-C data set. Bars of lighter color refer at the original models, while bars with darker colors at the models with a push–pull layer as the first layer (color figure online)

push–pull layer consistently obtains lower classification error.

The results that we achieved demonstrate that the use of the push–pull layer increases the average robustness of the concerned networks to various types of corruption of the input images. In order to learn effective representations and exploit the processing of the push–pull layer to improve generalization, a model is required to have adequately large capacity (i.e., number of learnable parameters). Smaller models, such as ResNet-20 or DenseNet-40-12, do not substantially benefit from the effect of the push–pull layer in the case of corruptions of the type *weather*, namely snow, frost, fog and spatter. In several cases, models with the push–pull layer achieve higher error than those with only convolutional layers. Larger models are able to better exploit the response map of the push–pull layer, especially for the *frost* corruption type. We draw similar observations from the results achieved on the *digital* corruptions, where the push–pull layer slightly improves the robustness of models of adequate size.

It is interesting to highlight the case of the *jpeg compression*, to which the results show a noticeable and systematic improvement of robustness of the networks that employ a push–pull layer. This result has very practical implications as jpeg is a widely used algorithm to compress image data. In real-world applications it is very likely that a classifier receives input images with varying compression level, and it is required to be robust to such corruptions.

We analyzed and compared the overall performance of the considered models with and without the push–pull layer with respect to the classification baseline results achieved by AlexNet. The choice of AlexNet is in line with the study reported in [17]. However, one can choose any other classifier as the baseline. For details about the configuration of the AlexNet model that we used for the experiments and

Table 2 Classification error achieved by the considered ResNet and DenseNet models on the CIFAR-C data set

Corruption	ResNet-20		ResNet-32		ResNet-44		ResNet-56		DenseNet-40-12		DenseNet-100-12		DenseNet-100-24	
	No PP	PP	No PP	PP	No PP	PP	No PP	PP	No PP	PP	No PP	PP	No PP	PP
Gaussian noise	57.41	49.9	58.38	47.81	54.5	48.43	56.72	44.92	68.67	52.31	56.19	45.2	57.83	42.83
Shot noise	45.75	39.98	47.31	37.12	42.93	38.11	44.02	35.74	56.25	42.45	44.01	34.38	44.12	32.33
Impulse noise	41.88	37.2	45.71	35.55	38.8	35.51	44.68	33.14	50.24	39.15	40.71	37.97	46.41	34.24
Speckle noise	41.34	37.55	43.47	33.96	39.22	35.42	39.72	33.58	51.15	39.74	39.14	31.78	38.68	29.32
Defocus blur	21.43	22.3	22.04	19.86	20.86	20.24	20.02	20.88	22.21	21.22	17.15	17.34	17.9	15.01
Glass blur	58.82	58.56	52.78	50.8	57.4	55.23	59.22	50.46	46.04	52.17	42.92	44.65	42.56	38.8
Motion blur	30.62	29.5	30.2	27.92	28.34	26.83	26.54	27.67	27.93	27.86	20.89	21.07	19.79	18.68
Zoom blur	27.54	29.68	30.33	26.92	28.06	25.89	26.73	26.58	30.62	27.54	22.94	22.19	24.26	18.98
Gaussian blur	31.32	33.24	33.94	30.12	32.83	29.93	30.88	31.22	36.07	30.7	28.58	28.23	33.6	25.25
Snow	25.06	25.59	23.19	22.6	23	23.66	21.99	22.32	21.73	25.01	16.87	18.34	14.41	15.67
Frost	31.39	31.29	30.86	26.37	28.02	26.69	29.08	27.28	28.59	30.25	23.39	22.48	20.16	18.72
Fog	16.56	17.58	16.08	15.45	14.98	14.98	14.55	15.93	17.19	17.12	13.11	12.88	10.66	11.38
Spatter	19.03	17.85	18.37	15.58	16.87	16.6	16.12	15.59	18.13	19.67	14.95	15.76	12.12	13.56
Brightness	9.73	10.34	9.2	8.98	8.63	8.54	8.3	8.68	8.6	9.49	6.3	6.85	5.4	5.97
Contrast	29.59	30.5	28.33	28.08	26.09	27.15	24.89	26.94	25.13	28.34	19.63	21.39	17.13	18.74
Elastic transf.	20.37	20.65	20.58	18.1	19.46	18.79	18.52	18.57	20.37	20.91	16.44	15.92	15.16	14.77
Pixelate	32.33	32.7	32.18	28.87	30.23	28.99	29.79	29.06	31.51	33.11	30.73	28.66	25.99	26.27
jpeg compr.	23.81	22.16	23.68	20.73	22.71	20.44	22.88	19.82	26.61	23.43	23.28	19.63	21.39	17.85
Saturate	11.52	11.7	10.9	11.24	9.94	9.8	9.56	9.9	11.63	11.86	8.25	8.43	7.15	7.61
Average error E	30.29	**29.38**	30.4	**26.63**	28.57	**26.91**	28.64	**26.33**	31.51	**29.07**	25.55	**23.85**	24.99	**21.37**

Best results are highlighted in bold

For each model configuration, we report the results obtained for every corruption type by the network version that employs a push–pull layer as first layer ('PP' column) and its convolutional only counterpart ('no PP' column). The results are grouped by corruption type (noise, blur, weather and digital)

the classification errors obtained for each corruption type, we refer the reader to "Appendix 2".

We report the mean corruption error mCE and the relative corruption error rCE achieved by the considered models in Table 3. A value of 100 indicates that there is no difference in performance between the concerned model and the baseline AlexNet model. Other values indicate the percentage measure of the average error with respect to that of AlexNet. For instance, 104 and 92 indicate that the error is 4% worse and 8% better, respectively, than that achieved by the baseline model. Thus, the ResNet-32-PP model achieves a corruption error that is 12% better than that of the ResNet-32 model. The mCE shows that some configurations of recent architectures, although achieving much lower classification error on clean data, are less robust to image corruptions than an AlexNet model. The ResNet-20 and DenseNet-40-12 models, for instance, achieved a mCE that indicates that the average corruption error on the CIFAR-C data set is, respectively, 4% and 8% higher than that obtained by AlexNet. The accuracy of a given model mainly depends on the specific architecture and amount of trainable parameters. When using models with enough

Table 3 Overall results obtained on the original CIFAR-10 test set (E_{clean} column) and on the CIFAR-C data set by the network models, with and without the proposed push–pull layer, that we considered for the experiments

Model	E_{clean}	E_{corr}	mCE	rCE
AlexNet (baseline)	13.87	29.08	100	100
ResNet-20	7.56	30.29	104	169
ResNet-20-PP	8.29	29.83	101	158
ResNet-32	7.29	30.4	104	171
ResNet-32-PP	7.15	26.63	92	145
ResNet-44	6.76	28.57	98	162
ResNet-44-PP	6.87	26.91	92	149
ResNet-52	6.64	28.64	97	161
ResNet-52-PP	7.01	26.23	91	144
DenseNet-40-12	6.38	31.51	108	187
DenseNet-40-12-PP	7.13	29.07	101	168
DenseNet-100-12	4.7	25.55	88	160
DenseNet-100-12-PP	5.04	23.85	82	144
DenseNet-100-24	3.88	24.99	84	156
DenseNet-100-24-PP	4.5	21.37	73	129

capacity (e.g., ResNet-52 and DenseNet-100-12), the mCE shows that generalization to corrupted data improves with respect to the performance achieved by AlexNet. For all the network configurations that we employed, the use of the push–pull layer as first layer of the architecture determines an average improvement of accuracy in presence of input corruptions, corresponding to a substantial decrease (up to 12% for ResNet-32-PP) of the mCE.

We observed that progressive improvements of the classification error achieved by the considered models with only convolutional layers on clean data did not correspond to similar improvements in generalization to corrupted data. The relative corruption error rCE measures the average gap between the classification error on clean and corrupted data. For all the tested models, the measured rCE indicates that the generalization capabilities of recent models are consistently worse than those of a much earlier AlexNet model. The improvement of the mCE obtained by models with only convolutional layers is mostly due to the increase in classification accuracy on clean data and model capacity of successively published architectures, rather than to an improvement of generalization [17].

This does not hold when using a push–pull layer in the network architecture. In this case, on the one hand, we noticed a very small increase in the classification error on clean data, but on the other we recorded a substantial and systematic improvement of the mCE and rCE. Although the generalization capabilities of a classifier to corrupted input data depend on the particular architecture and its amount of parameters, the push–pull layer consistently contributes to achieve a lower corruption error and embues the concerned model with an intrinsic improved robustness to various input corruptions. The push–pull layer contributes to reduce the rCE, which indicates better generalization to corrupted data.

4.3 Sensitivity to push–pull parameters

We performed an evaluation of the sensitivity of the classification error with respect to variations of the parameters of the push–pull layer, namely the upsampling factor h and the inhibition strength α. In Table 4, we report the results that we achieved with several ResNet-14-PP models, for which we configured push–pull layers with different h and α parameter values. We tested the performance of these models on the CIFAR-10 data set images, which we corrupted by means of added Gaussian noise of increasing severity. The first row of the tables reports the results of the ResNet-14 model without the push–pull layer.

We observed that, despite the particular configuration of the hyper-parameters h and α, the push–pull layer contributes to improve, often substantially, the robustness to added Gaussian noise of increasing variance. Models with the push–pull layer usually achieve a slightly lower accuracy on the clean test images, while having higher robustness to corruptions. The sensitivity analysis on Gaussian noise corruption shows that the configuration of the push–pull layer makes the concerned models more or less robust to a certain corruption. We configured, for the experiments discussed in the previous sections, the push–pull layer with $h = 2$ and $\alpha = 1$ as these values contribute to achieve on average good performance on a number of corruption types. However, one may optimize the model for a certain type of input perturbation, which depends on the particular application at hand.

Table 4 Sensitivity analysis of the classification error with respect to changes of the configuration parameters of the push–pull layer in a ResNet-14 model

Sensitivity analysis of ResNet-PP w.r.t. inhibition parameters							
		Severity of Gaussian noise (σ^2)					
h	α	0	0.0001	0.0005	0.001	0.005	0.01
–	–	4.09	5.18	10.47	18.76	59.53	77.46
1	0.5	**3.99**	**4.81**	9.5	17.1	57.2	72.51
1	1	4.17	4.86	9.58	16.62	55.95	71.17
1	1.5	4.19	4.97	10.11	17.85	60.79	77.32
1.5	0.5	4.24	4.98	8.76	14.26	45.92	62.56
1.5	1	4.16	5.17	9.18	14.64	47.97	65.58
1.5	1.5	4.33	4.91	8.32	12.82	42.41	59.82
2	0.5	4.38	4.72	8.54	13.18	41.15	**58.86**
2	1	4.55	5.97	10.1	14.94	44.56	63.91
2	1.5	4.38	4.97	**7.98**	**11.98**	**41.11**	63.17

In bold, we report the best result for each severity of Gaussian noise added to the CIFAR-10 test set

4.4 Learning the inhibition strength α

The value of the α parameter of the proposed push–pull model can be learned during training. We report in Table 5 results achieved by models in which the value of α is trained (subscript α is present in the model name), compared to those of models with fixed value of $\alpha = 1$ and

Table 5 Comparison of the classification and corruption error achieved by learning or not the value of the inhibition strength α

Learned α parameter		
	E_{clean}	E_{corr}
ResNet-20	7.56	30.29
ResNet-20-PP	8.29	29.38
ResNet-20-PP$_\alpha$	7.89	31.09
ResNet-56	6.64	28.64
ResNet-56-PP	7.01	26.23
ResNet-56-PP$_\alpha$	6.48	26.82

without push–pull inhibition. In the case of ResNet-20, we did not observe an improvement of the robustness of the network with push–pull layers and learned inhibition strength. In the case of ResNet-56, the learned inhibition strength contributes to a consistent improvement of the robustness also slightly reducing the classification error on clean data. However, we observed that the optimization process becomes more complicated, with slower convergence, and requires longer training time when the value of the α parameter is learned. Hence, it is worth to consider further future analyses to study the effects of learning the inhibition strength on the configuration of the training process, i.e., the learning rate schedule, batch size, etc.

5 Discussion

We demonstrated that the use of a push–pull layer as substitute of the first convolutional layer of state-of-the-art CNNs contributes to a substantial increase in robustness of the networks to corruption of the input images. This is attributable to the capability of a push–pull kernel to detect a feature of interest more robustly than a convolutional kernel also when the input data is heavily corrupted. Being robust to different kinds of corruption of the input, for instance in the case of low illumination or adverse weather conditions that disturb our visual perception, is a key property of the visual system of the brain. Inhibition phenomena at different levels of the visual system hierarchy are known to play a key role in such mechanism [2].

The design of the push–pull layer is inspired by the functions of some neurons in the early part of the visual system of the brain, which exhibit a response inhibition phenomenon known as push–pull inhibition. In this light, the implementation of the push–pull layer strengthens the relation between the processing of visual information that is performed inside a CNN and that in the human system of the brain. Indeed, the hierarchical computation of features with increasing semantic value in the CNN architectures resemble the layered organization of the visual system. In this work, we augmented actual convolutional neural networks with an explicit implementation of inhibition of visual responses as it is known to happen in area V1 of the visual cortex [21].

In the experiments, we used a fixed value of the inhibition strength α for all the kernels in a push–pull layer. It is, however, known from neuro-physiological studies that not all the neurons in area V1 of the visual system of the brain exhibit push–pull inhibition properties. Furthermore, those neurons that have an inhibitory component do not all actuate it with the same intensity [42]. This behavior can be implemented in the proposed push–pull layer by training the value α_i of the inhibition strength of the i-th kernel in a push–pull layer. In this way, only few kernels are expected to implement inhibition functions, according to what is known to happen in the visual system.

In principle, one can deploy the push–pull layer at any level into a CNN architecture, as it is designed as a substitute of a convolutional layer. However, neuro-physiological studies recorded the functions of push–pull inhibition only in the early parts of the visual cortex, up to the area V1. In a CNN, these areas can be related to the first group of convolutional layer, e.g., the first residual block of ResNet or the first dense block of DenseNet. However, it is expected that deploying the proposed push–pull layer inside a residual or dense block changes the learning dynamics of the classification models, making the optimization process more difficult. Thus, further studies are needed to employ the push–pull layer at deeper layers in CNN architectures.

6 Conclusions

We proposed a novel push–pull layer for CNN architectures, which increases the robustness of existing networks to various corruptions of the input images. Its parameters can be trained by error backpropagation, similarly to those of convolutional layers. This allows to use the push–pull layer as a substitute of any convolutional layer in a CNN architecture.

We validated the effectiveness of the push–pull layer by deploying it in state-of-the-art CNN architectures, by substituting the first convolutional layer. The results that we achieved using LeNet on the MNIST data set, and ResNet and DenseNet models on corrupted versions of the CIFAR data set, namely the CIFAR-C benchmark data set, demonstrate that the push–pull layer considerably increases the robustness of existing networks to input image corruptions. Furthermore, the use of the push–pull layer as first layer of any CNN, replacing the first convolutional layers, guarantees a systematic improvement of generalization capabilities of the network, which we measured by a substantial reduction of the relative corruption error between performance on clean and corrupted data. We released the code and trained models at the url http://github.com/nicstrisc/Push-Pull-CNN-layer.

Funding This study was partially funded by the European Commission H2020 program under project TrimBot2020 (Grant Number 688007).

Compliance with ethical standards

Conflict of interest The authors declare that they have no further conflict of interest.

Appendix 1: Brain-inspired model design

The design of the proposed push–pull block is inspired by neuro-physiological evidence of the presence of a particular form of inhibition, called push–pull inhibition, in the visual system of the brain.

In general, inhibition is the phenomenon of suppression of the response of certain neural receptive fields by means of the action of receptive fields with opposite polarity. From neuro-physiological studies of the visual system of the brain, there is evidence that neurons exhibit various forms of inhibition. For instance, end-stopped cells are characterized by an inhibition mechanism that increases their selectivity to line-ending patterns [7]. In the case of lateral inhibition, the response of a certain neuron suppresses the responses of its neighboring neurons. Lateral inhibition inspired the design of the Local Response Normalization technique in CNNs, which increased the generalization results of AlexNet [22]. Center-surround inhibition is known to increase the detection rate of patterns of interest by suppression of texture in their surroundings, and has been shown to be effective in image processing [15].

Neurons that exhibit push–pull inhibition are composed of one receptive field that is excited by a certain positive stimulus (push) and one that is excited by its negative counterpart (pull). In practice, the negative receptive field is larger than the positive one and suppresses its response [26, 27]. The effect of push–pull inhibition is to increase the selectivity of neurons for stimuli for which they are tuned, even when they are corrupted by noise [11].

Appendix 2: AlexNet on CIFAR-C: baseline results

In Fig. 8, we depict the AlexNet architecture that we used for the baseline experiments on the CIFAR-10 data set. The concerned model is the result of a modification of the ImageNet version of AlexNet, that is configured to work with images from the CIFAR data set. The main differences with respect to the ImageNet version of AlexNet is that in this implementation, the size of all the convolutional kernels is 3×3 pixels and the input size of the first layer of the fully connected network is 1024. Furthermore, we set the stride of the first convolution *conv1* equal to 2 (instead of 4 as in the ImageNet model).

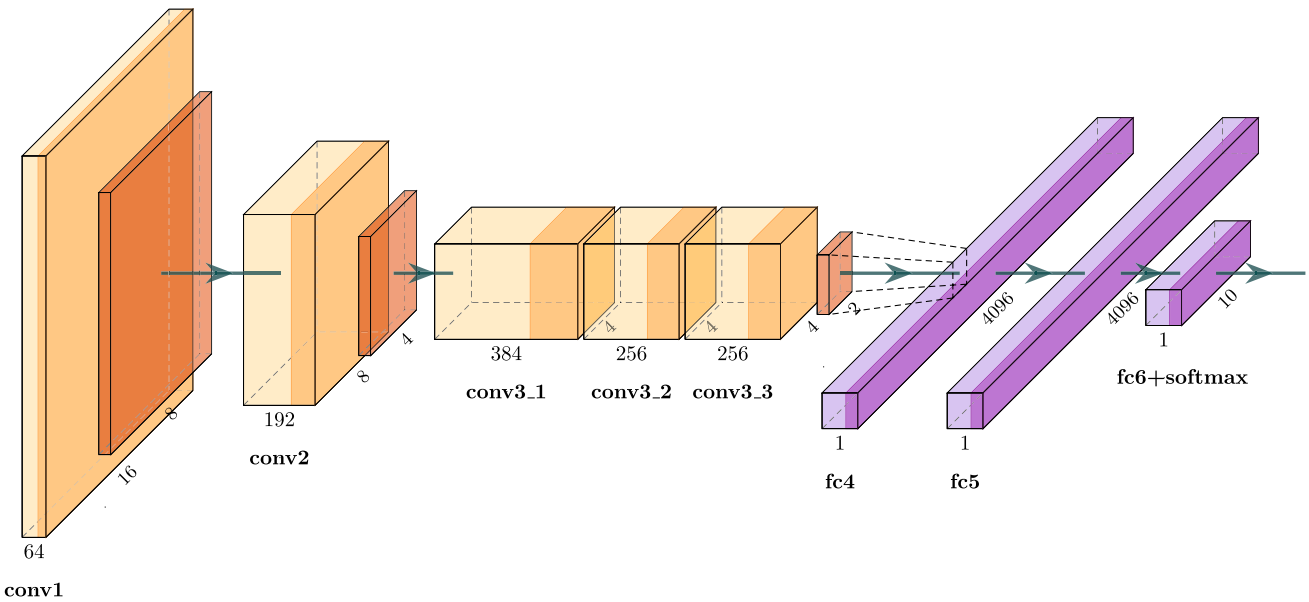

Fig. 8 AlexNet architecture used as the baseline model for the analysis of results on the CIFAR-C data set. Light yellow indicates a convolution, while darker yellow is a ReLU function. Similarly, light purple is a linear layer while the darker purple box indicates the corresponding ReLU function. The max-pooling operation is represented with the orange box (color figure online)

The used AlexNet model achieved a classification error on clean data of 13.87. In the following we report the values of the AlexNet classification error (as percentage) on the corrupted CIFAR-C test sets, which we used to normalize the corruption errors $CE^{M}_{c,s}$—Gaussian noise: 38.44, shot noise: 31.82, impulse noise: 46.13, speckle noise: 31.26, defocus blur: 23.53, glass blur: 40.97, motion blur: 29.32, zoom blur: 26.89, Gaussian blur: 27.78, snow: 26.94, frost: 28.26, fog: 30.31, spatter: 25.19, brightness: 17.84, contrast: 46.77, elastic transformation: 22.25, pixelate: 18.36, jpeg compression: 18.48, saturate: 22.15.

References

1. Akhtar N, Mian A (2018) Threat of adversarial attacks on deep learning in computer vision: a survey. IEEE Access 6:14410–14430. https://doi.org/10.1109/access.2018.2807385
2. Alitto HJ, Dan Y (2010) Function of inhibition in visual cortical processing. Curr Opin Neurobiol 20(3):340–346. https://doi.org/10.1016/j.conb.2010.02.012
3. Azzopardi G, Petkov N (2012) A CORF computational model of a simple cell that relies on LGN input outperforms the Gabor function model. Biol Cybern 106(3):177–189. https://doi.org/10.1007/s00422-012-0486-6
4. Azzopardi G, Rodríguez-Sánchez A, Piater J, Petkov N (2014) A push–pull corf model of a simple cell with antiphase inhibition improves snr and contour detection. PLoS ONE 9(7):e98424. https://doi.org/10.1371/journal.pone.0098424
5. Azzopardi G, Strisciuglio N, Vento M, Petkov N (2015) Trainable cosfire filters for vessel delineation with application to retinal images. Med Image Anal 19(1):46–57. https://doi.org/10.1016/j.media.2014.08.002
6. Badrinarayanan V, Kendall A, Cipolla R (2015) Segnet: a deep convolutional encoder-decoder architecture for image segmentation. CoRR arXiv:1511.00561
7. Bolz J, Gilbert CD (1986) Generation of end-inhibition in the visual cortex via interlaminar connections. Nature 320(6060):362–365
8. Carlini N, Wagner DA (2016) Towards evaluating the robustness of neural networks. CoRR arXiv:abs/1608.04644
9. Cohen TS, Welling M (2016) Steerable cnns. CoRR arXiv:abs/1612.08498
10. Dodge S, Karam L (2017) A study and comparison of human and deep learning recognition performance under visual distortions. In: 2017 26th international conference on computer communication and networks (ICCCN), pp 1–7. https://doi.org/10.1109/ICCCN.2017.8038465
11. Freeman TC, Durand S, Kiper DC, Carandini M (2002) Suppression without inhibition in visual cortex. Neuron 35(4):759–771. https://doi.org/10.1016/S0896-6273(02)00819-X
12. Fukushima K (1980) Neocognitron: a self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biol Cybern 36(4):193–202. https://doi.org/10.1007/BF00344251
13. Geirhos R, Temme CRM, Rauber J, Schütt HH, Bethge M, Wichmann FA (2018) Generalisation in humans and deep neural networks. In: Advances in neural information processing systems 31 (NeurIPS2018), pp 7538–7550
14. Goodfellow I, Shlens J, Szegedy C (2015) Explaining and harnessing adversarial examples. In: International conference on learning representations. arXiv:1412.6572
15. Grigorescu C, Petkov N, Westenberg M (2004) Contour and boundary detection improved by surround suppression of texture edges. Image Vis Comput 22(8):609–622. https://doi.org/10.1016/j.imavis.2003.12.004
16. He K, Zhang X, Ren S, Sun J (2015) Deep residual learning for image recognition. CoRR arXiv:abs/1512.03385
17. Hendrycks D, Dietterich T (2019) Benchmarking neural network robustness to common corruptions and perturbations. In: Proceedings of the international conference on learning representations
18. Huang G, Liu Z, van der Maaten L, Weinberger KQ (2017) Densely connected convolutional networks. In: 2017 IEEE conference on computer vision and pattern recognition (CVPR), pp 2261–2269
19. Hubel D, Wiesel T (1962) Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. J Physiol-Lond 160(1):106–154
20. Hui TW, Tang X, Loy CC (2018) Liteflownet: a lightweight convolutional neural network for optical flow estimation. In: Proceedings of IEEE conference on computer vision and pattern recognition (CVPR), pp 8981–8989
21. Kremkow J, Perrinet LU, Monier C, Alonso JM, Aertsen A, Frégnac Y, Masson GS (2016) Push–pull receptive field organization and synaptic depression: mechanisms for reliably encoding naturalistic stimuli in v1. Front Neural Circuits 10:37. https://doi.org/10.3389/fncir.2016.00037
22. Krizhevsky A, Sutskever I (2012) Imagenet classification with deep convolutional neural networks. In: Pereira F, Burges CJC, Bottou L, Weinberger KQ (eds) Advances in neural information processing systems 25. Curran Associates, Inc., New York, pp 1097–1105
23. Kurakin A, Goodfellow IJ, Bengio S (2016) Adversarial examples in the physical world. CoRR arXiv:abs/1607.02533. URL http://dblp.uni-trier.de/db/journals/corr/corr1607.html#KurakinGB16
24. LeCun Y, Haffner P, Bottou L, Bengio Y (1999) Object recognition with gradient-based learning. In: Forsyth D (ed) Feature grouping. Springer, Berlin
25. Lee CY, Xie S, Gallagher P, Zhang Z, Tu Z (2015) Deeply-supervised nets. In: Proceedings of the eighteenth international conference on artificial intelligence and statistics, proceedings of machine learning research, vol 38, pp 562–570. PMLR
26. Li Y, Ma WP, Li LY, Ibrahim LA, Wang SZ, Tao HW (2012) Broadening of inhibitory tuning underlies contrast-dependent sharpening of orientation selectivity in mouse visual cortex. J Neurosci 32(46):16466–16477. https://doi.org/10.1523/JNEUROSCI.3221-12.2012
27. Liu BH, tang Li YT, Ma WP, Pan CJ, Zhang LI, Tao HW (2011) Broad inhibition sharpens orientation selectivity by expanding input dynamic range in mouse simple cells. Neuron 71(3):542–554. https://doi.org/10.1016/j.neuron.2011.06.017
28. Lu J, Sibai H, Fabry E, Forsyth DA (2017) Standard detectors aren't (currently) fooled by physical adversarial stop signs. CoRR arXiv:abs/1710.03337
29. Madry A, Makelov A, Schmidt L, Tsipras D, Vladu A (2018) Towards deep learning models resistant to adversarial attacks. CoRR arXiv:abs/1706.06083
30. Marĉelja S (1980) Mathematical description of the responses of simple cortical cells∗. J Opt Soc Am 70(11):1297–1300. https://doi.org/10.1364/JOSA.70.001297
31. Metzen JH, Genewein T, Fischer V, Bischoff B (2017) On detecting adversarial perturbations. In: Proceedings of 5th

international conference on learning representations (ICLR). arXiv:1702.04267
32. Moosavi-Dezfooli S, Fawzi A, Frossard P (2016) Deepfool: a simple and accurate method to fool deep neural networks. In: CVPR. IEEE Computer Society, pp 2574–2582
33. Moosavi-Dezfooli S, Fawzi A, Fawzi O, Frossard P (2017) Universal adversarial perturbations. In: CVPR. IEEE Computer Society, pp 86–94
34. Papernot N, McDaniel PD, Wu X, Jha S, Swami A (2015) Distillation as a defense to adversarial perturbations against deep neural networks. CoRR arXiv:abs/1511.04508
35. Simonyan K, Zisserman A (2014) Very deep convolutional networks for large-scale image recognition. CoRR arXiv:abs/1409.1556
36. Song X, Zhao X, Hu H, Fang L (2018) Edgestereo: a context integrated residual pyramid network for stereo matching. CoRR arXiv:abs/1803.05196
37. Strisciuglio N, Petkov N (2017) Delineation of line patterns in images using b-cosfire filters. In: IWOBI, pp 1–6. https://doi.org/10.1109/IWOBI.2017.7985538
38. Strisciuglio N, Azzopardi G, Petkov N (2019) Brain-inspired robust delineation operator. In: Computer Vision—ECCV 2018 Workshops, pp 555–565
39. Strisciuglio N, Azzopardi G, Petkov N (2019) Robust inhibition-augmented operator for delineation of curvilinear structures. IEEE Trans Image Process 28(12):5852–5866. https://doi.org/10.1109/TIP.2019.2922096
40. Szegedy C, Inc G, Zaremba W, Sutskever I, Inc G, Bruna J, Erhan D, Inc G, Goodfellow I, Fergus R (2014) Intriguing properties of neural networks. In: In ICLR
41. Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A (2015) Going deeper with convolutions. In: Computer vision and pattern recognition (CVPR). arXiv:1409.4842
42. Taylor MM, Sedigh-Sarvestani M, Vigeland L, Palmer LA, Contreras D (2018) Inhibition in simple cell receptive fields is broad and off-subregion biased. J Neurosci 38(3):595–612. https://doi.org/10.1523/JNEUROSCI.2099-17.2017
43. Temel D, Kwon G, Prabhuhankar M, AlRegib G (2017) CURE-TSR: challenging unreal and real environments for traffic sign recognition. In: Advances in neural information processing systems (NIPS) machine learning for intelligent transportations systems workshop
44. Temel D, Lee J, AlRegib G (2018) CURE-OR: challenging unreal and real environments for object recognition. CoRR arXiv:abs/1810.08293
45. Vasiljevic I, Chakrabarti A, Shakhnarovich G (2016) Examining the impact of blur on recognition by convolutional networks. CoRR arXiv:abs/1611.05760
46. Weiler M, Hamprecht FA, Storath M (2017) Learning steerable filters for rotation equivariant cnns. CoRR arXiv:abs/1711.07289
47. Worrall DE, Garbin SJ, Turmukhambetov D, Brostow GJ (2016) Harmonic networks: deep translation and rotation equivariance. CoRR arXiv:abs/1612.04642
48. Zagoruyko S, Komodakis N (2016) Wide residual networks. In: BMVC
49. Zheng S, Song Y, Leung T, Goodfellow I (2016) Improving the robustness of deep neural networks via stability training. In: 2016 IEEE conference on computer vision and pattern recognition (CVPR), pp 4480–4488. https://doi.org/10.1109/CVPR.2016.485
50. Zoumpourlis G, Doumanoglou A, Vretos N, Daras P (2017) Non-linear convolution filters for cnn-based learning. In: IEEE international conference on computer vision, ICCV 2017, Venice, Italy, October 22–29, 2017, pp 4771–4779. https://doi.org/10.1109/ICCV.2017.510