How deep convolutional neural networks lose spatial information with training

IOP Publishing Mach. Learn.: Sci. Technol. 4 (2023) 045026 https://doi.org/10.1088/2632-2153/ad092c

MACHINE LEARNING
Science and Technology

OPEN ACCESS

RECEIVED
14 June 2023

REVISED
9 October 2023

ACCEPTED FOR PUBLICATION
2 November 2023

PUBLISHED
9 November 2023

PAPER

How deep convolutional neural networks lose spatial information with training

Umberto M Tomasini*, Leonardo Petrini, Francesco Cagnetta and Matthieu Wyart

Institute of Physics, École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland
* Author to whom any correspondence should be addressed.

E-mail: umberto.tomasini@epfl.ch

Keywords: deep learning theory, convolutional neural networks, curse of dimensionality, representation learning, feature learning, learning invariants

Abstract

A central question of machine learning is how deep nets manage to learn tasks in high dimensions. An appealing hypothesis is that they achieve this feat by building a representation of the data where information irrelevant to the task is lost. For image datasets, this view is supported by the observation that after (and not before) training, the neural representation becomes less and less sensitive to diffeomorphisms acting on images as the signal propagates through the network. This loss of sensitivity correlates with performance and surprisingly correlates with a *gain* of sensitivity to white noise acquired during training. Which are the mechanisms learned by convolutional neural networks (CNNs) responsible for the these phenomena? In particular, why is the sensitivity to noise heightened with training? Our approach consists of two steps. (1) Analyzing the layer-wise representations of trained CNNs, we disentangle the role of spatial pooling in contrast to channel pooling in decreasing their sensitivity to image diffeomorphisms while increasing their sensitivity to noise. (2) We introduce model scale-detection tasks, which qualitatively reproduce the phenomena reported in our empirical analysis. In these models we can assess quantitatively how spatial pooling affects these sensitivities. We find that the increased sensitivity to noise observed in deep ReLU networks is a mechanistic consequence of the perturbing noise piling up during spatial pooling, after being rectified by ReLU units. Using odd activation functions like tanh drastically reduces the CNNs' sensitivity to noise.

1. Introduction

Deep learning algorithms can be successfully trained to solve a large variety of tasks [1–5], often revolving around classifying data in high-dimensional spaces. If there was little structure in such tasks, the learning procedure would be cursed by the dimension d of these spaces. Indeed, generic tasks like regression of a continuous function [6] rely on sampling neighboring points in the data space to achieve good performance. Since the typical distance δ between neighbors scales with the number of training points P as $\delta \sim P^{-1/d}$, a number of points exponential in d would be needed to reach low δ and consequently good performance. For a benchmark dataset like Imagenet, the effective dimension is ≈ 50 [7], implying that a number of points of order $e^{50} \approx 10^{20}$ would be required for efficient sampling of the data space. However, modern machine learning models achieve good performances with much fewer training data [8, 9]. Consequently, learnable tasks on real datasets must have a specific internal structure that can be learned with fewer examples. It has been then hypothesized that the effectiveness of deep learning lies in its ability to build 'good' representations of this internal structure, which are insensitive to aspects of the data not related to the task [10–12], thus effectively reducing the dimensionality of the problem.

In the context of image classification, [13, 14] proposed that neural networks lose irrelevant information by learning representations that are insensitive to small deformations of the input, also called

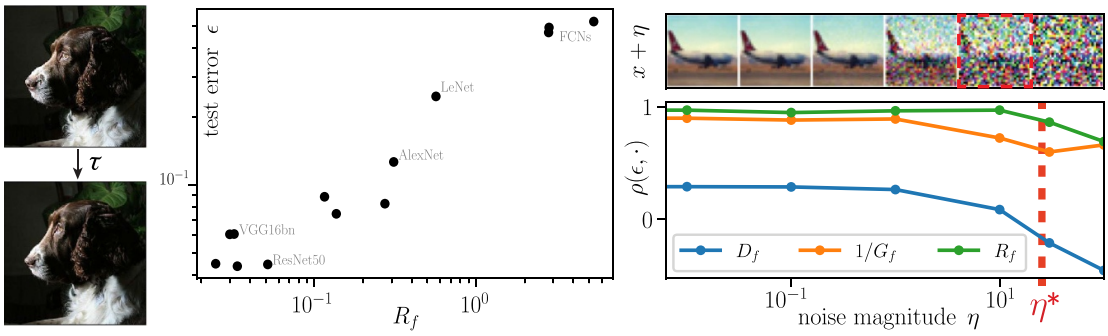

Figure 1. Left: example of a random diffeomorphism τ applied to an image. Center: test error vs relative sensitivity to diffeomorphisms of the predictor for a set of networks trained on CIFAR10, adapted from [15]. Right: correlation coefficient between test error ε and D_f, G_f and R_f when training different architectures on noisy CIFAR10, $\rho(\epsilon, X) = \mathrm{Cov}(\log\epsilon, \log X)/\sqrt{\mathrm{Var}(\log\epsilon)\mathrm{Var}(\log X)}$. Increasing noise magnitudes are shown on the x-axis and $\eta^* = \mathbb{E}_{\tau,x}\|\tau(x) - x\|^2$ is the one used for the computation of G_f. Samples of a noisy CIFAR10 datum are shown on top. Notice that D_f and particularly R_f are positively correlated with ϵ, whilst G_f is negatively correlated with ϵ. The corresponding scatter plots are in figure C3 in appendix C.

diffeomorphisms. These representations would simplify the task by effectively reducing its dimensionality. This idea was tested in modern deep networks by [15], who introduced the following measures

$$D_f = \frac{\mathbb{E}_{x,\tau}\|f(\tau(x)) - f(x)\|^2}{\mathbb{E}_{x_1,x_2}\|f(x_1) - f(x_2)\|^2}, \qquad G_f = \frac{\mathbb{E}_{x,\eta}\|f(x+\eta) - f(x)\|^2}{\mathbb{E}_{x_1,x_2}\|f(x_1) - f(x_2)\|^2}, \qquad R_f = \frac{D_f}{G_f}, \tag{1}$$

to probe the sensitivity of a function f—either the output or an internal representation of a trained network—to random diffeomorphisms τ of x (see example in figure 1, left), to large white noise perturbations η of magnitude $\|\tau(x) - x\|$, and in relative terms, respectively. Here the input images x, x_1, and x_2 are sampled uniformly from the test set. In particular, the test error of trained networks is correlated with D_f when f is the network output. Less intuitively, the test error is anti-correlated with the sensitivity to white noise G_f. Overall, it is the relative sensitivity R_f which correlates best with the error (figure 1, middle, adapted from [15]), suggesting that the best networks are the ones less sensitive to diffeomorphisms, in relative terms. This correlation is learned over training—as it is not seen at initialization—and built up layer by layer [15]. These phenomena are not simply due to benchmark data being noiseless, as they persist when input images are corrupted by some small noise (figure 1, right). In our work we aim to understand which are the mechanisms allowing convolutional neural networks (CNNs) to become insensitive to diffeomorphisms while increasing their sensitivity to random noise.

Operations that grant insensitivity to diffeomorphisms in a deep network have been identified previously (e.g. [16, section 9.3], sketched in figure 2). The first, *spatial* pooling, integrates local patches within the image, thus losing the exact location of its features. The second, *channel* pooling, requires the interaction of different channels, which allows the network to become insensitive to any local transformation by properly learning filters that are transformed versions of one another. However, it is not clear whether these operations are learned by deep networks and how they conspire in building good representations insensitive to diffeomorphisms. Here, we address this question through a two-fold approach. First, we empirically isolate the distinct impact of spatial pooling versus channel pooling in mitigating sensitivity to diffeomorphisms while enhancing sensitivity to noise. Second, we introduce model scale-detection tasks, which qualitatively replicate the phenomena observed in our empirical analysis. These models allow us to quantitatively evaluate the impact of spatial pooling on sensitivity to both diffeomorphisms and noise. As a result of our theoretical study, we understand that the sensitivity to noise increases with training due to the cumulative effect of perturbing noise following the rectifying action of the ReLU activation function, explaining the empirical observations of [15]. Below there is a detailed list of our contributions.

1.1. Our contributions

- In section 2 we empirically observe that deep networks trained on CIFAR10 and ImageNet reduce their sensitivity to diffeomorphisms with training. We argue that both spatial and channel pooling are learned and we disentangle their role. More specifically, our experiments reveal the significant contribution of spatial pooling in decreasing the sensitivity to diffeomorphisms.
- Since spatial pooling plays a relevant role in making the trained CNNs less sensitive to diffeomorphisms, in section 3 we introduce idealized scale-detection tasks to isolate its contribution. In these tasks, data are

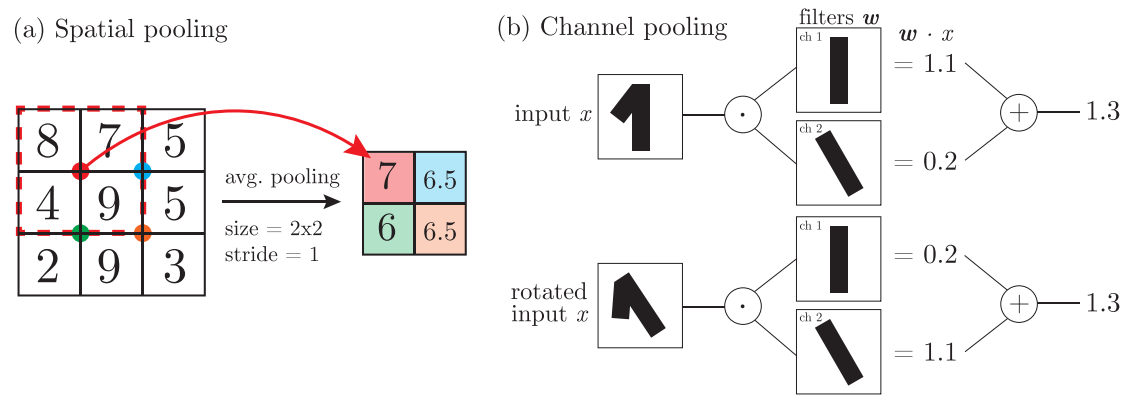

Figure 2. Spatial vs. channel pooling. (a) Spatial average pooling (size 2×2, stride 1) computed on a representation of size 3×3. One can notice that nearby pixel variations are smaller after pooling. (b) If the filters of different channels are identical up to e.g. a rotation of angle θ, then, averaging the output of the application of such filters makes the result invariant to input rotations of θ. This averaging is an example of channel pooling.

made of two active pixels and classified according to their distance: if they are close the label is $+1$, otherwise -1. We observe that the solutions learned by deep CNNs perform spatial pooling, as we expected: if inside the learned spatial pooling filter there are both active pixels, the label is 1, otherwise -1. Moreover, we find the same correlations between test error and sensitivities of trained networks as found in [15]. In addition, the neural networks that perform the best on real data tend to be the best on these tasks.

- Finally, in section 4 we theoretically analyze how simple CNNs with ReLU activation functions, made by stacking convolutional layers with filter size F and stride s, learn the idealized tasks, and we ultimately show why the sensitivity to noise increases with training. In particular, we find that the trained networks perform spatial pooling for most of their layers. Based on a spatial pooling solution, we show and verify empirically that the sensitivities D_k and G_k of the kth hidden layer follow $G_k \sim A_k$ and $D_k \sim A_k^{-\alpha_s}$, where A_k is the effective receptive field size and $\alpha_s = 2$ if there is no stride, $\alpha_s = 1$ otherwise. We remark that replacing the ReLU activation function with an odd sigmoidal one such as tanh results, as shown in appendix B in $G_k \sim C$, independently of the layer k. Therefore, the increase in the sensitivity to random transformations is a mechanistic consequence of performing spatial pooling with ReLU activation functions.

The code and details for reproducing experiments are available online at github.com/leonardopetrini/relativestability/blob/main/experiments.md.

1.2. Related work

In the neuroscience literature, the understanding of the relevance of pooling in building invariant representations dates back to the pioneering work of [17]. By studying the cat visual cortex, they identified two different kinds of neurons: simple cells responding to e.g. edges at specific angles and complex cells that *pool* the response of simple cells and detect edges regardless of their position or orientation in the receptive field. More recent accounts of the importance of learning invariant representations in the visual cortex can be found in [18–20].

In the context of artificial neural networks, layers jointly performing spatial pooling and strides have been introduced with the early CNNs of [21], following the intuition that local averaging and subsampling would reduce the sensitivity to small input shifts. Reference [22] investigated the role of spatial pooling and showed empirically that networks with and without pooling layers converge to similar deformation sensitivity, suggesting that spatial pooling can be learned in deep networks. In our work, we further expand in this direction by jointly studying diffeomorphisms and noise sensitivity and proposing a theory of spatial pooling for a simple task.

In the context of tasks being invariant, for example to rotations, CNNs were developed to have internal representations equivariant to these transformations [23–30]. Enforcing pooling operations in these networks would presumably lead to networks quite stable to diffeomorphisms. Here instead we are concerned with understanding how approximate invariance to diffeomorphisms is learned while training standard CNNs, without imposing it at the initialization in the architecture. Indeed, most state-of-the-art image classification CNNs do not enforce such invariance, supporting that it is better to let the network tune its pooling operation.

The depth-wise loss of irrelevant information in deep networks has been investigated by means of the information bottleneck framework [11, 31] and the intrinsic dimension of the network's internal representations [10, 12]. However, these works do not specify what is the irrelevant information to be disregarded, nor the mechanisms involved in such a process.

The stability of trained networks to noise is extensively studied in the context of adversarial robustness [32–38]. Notice that our work differs from this literature by the fact that we consider typical perturbations instead of worst-case ones.

2. Empirical observations on real data

In this section, we analyze the learned weights of deep CNNs trained on CIFAR10 and ImageNet, to understand how they build representations insensitive to diffeomorphisms. All experiments are performed in PyTorch [39]. The code with the instructions on how to reproduce experiments is found here: github.com/leonardopetrini/relativestability/blob/main/experiments.md. Our primary focus is on evaluating the performance of existing neural network architectures. For this reason, we used the default, off-the-shelf values for parameters such as depth and number of channels. We aimed for a unified training process that could effectively minimize training errors across diverse architectures within a reasonable time frame. For CIFAR10 experiments, after a grid search among different optimizers, learning rates, schedulers, and momentum values we found the following configuration to be the most effective for this goal: ADAM optimizer with learning rate $= 0.1$ for fully connected networks while for CNNs we use SGD, learning rate $= 0.1$ and momentum $= 0.9$. In the latter case, the learning rate follows a cosine annealing scheduling. The networks are trained on the cross-entropy loss, with a batch size of 128 and for 250 epochs. Early stopping at the best validation error is performed for selecting the networks to study. During training, we employ standard data augmentation consisting of random translations and horizontal flips of the input images. All results are averaged when training on 5 or more different network initializations. For ImageNet, we used pre-trained models from PyTorch, `torchvision.models`.

Our analysis of the learned representations builds on two premises, the first being that insensitivity is built layer by layer in the network, as shown in figure 3. Hence, we focus on how each of the layers of a deep network contributes to creating an insensitive representation. More specifically, let us denote with $f_k(x)$ the internal representation of an input x at the kth layer of the network. The entries of f_k have three indices, one for the channel c and two for the spatial location (i, j). The relation between f_k and f_{k-1} is the following,

$$[f_k(x)]_{c;i,j} = \phi\left(b_c^k + \sum_{c'=1}^{H_{k-1}} \boldsymbol{w}_{c,c'}^k \cdot \boldsymbol{p}_{i,j}\left([f_{k-1}(x)]_{c'}\right)\right) \tag{2}$$

for all $c = 1, \ldots, H_k$, where: H_k denotes the number of channels at the kth layer; b_c^k and $\boldsymbol{w}_{c,c'}^k$ the biases and *filters* of the kth layer; each filter $\boldsymbol{w}_{c,c'}^k$ is a $F \times F$ matrix with F the filter size; $\boldsymbol{p}_{i,j}\left([f_{k-1}(x)]_{c'}\right)$ denotes a $F \times F$-dimensional patch of $[f_{k-1}(x)]_{c'}$ centered at (i, j); ϕ the activation function. The second premise is that a general diffeomorphism can be represented as a displacement field over the image, which indicates how each pixel moves in the transformation. Locally, this displacement field can be decomposed into a constant term and a linear part: the former corresponds to local translations, the latter to stretchings, rotations, and shears[1]. We generate the random diffeomorphisms according to the implementation proposed in [15]. Specifically, the displacement field is sampled uniformly at random from all the displacement fields compatible with a fixed average displacement. The spatial frequency content of the sampled diffeomorphisms is cut off at a given high-frequency threshold to ensure the bijectivity of the transformation (see [15] for further details). Each pixel of the input image grid is displaced according to the aforementioned sampling. Generically it is mapped outside the original grid. To project back to the original grid, we employ a bi-linear interpolation.

Representations insensitive to translations via spatial pooling. Due to weight sharing, i.e. the fact that the same filter $\boldsymbol{w}_{c,c'}^k$ is applied to all the local patches (i, j) of the representation, the output of a convolutional layer is *equivariant* to translations by construction: a shift of the input is equivalent to a shift of the output. To achieve an *invariant* representation it suffices to sum up the spatial entries of f_k—an operation called

[1] The displacement field around a pixel (u_0, v_0) is approximated as $\tau(u, v) \simeq \tau(u_0, v_0) + J(u_0, v_0)[u - u_0, v - v_0]^T$, where $\tau(u_0, v_0)$ corresponds to translations and J is the Jacobian matrix of τ whose trace, antisymmetric and symmetric traceless parts correspond to stretchings, rotations and shears, respectively.

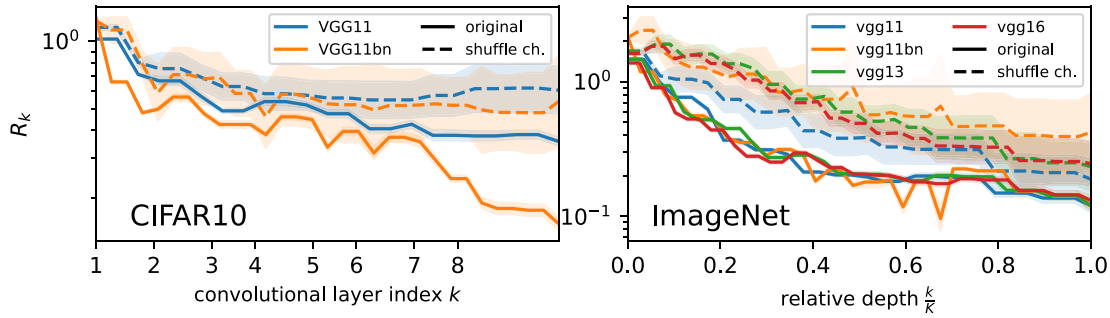

Figure 3. Relative sensitivity R_k as a function of depth for VGG architectures trained on CIFAR10 (left) and ImageNet (right). Solid lines refer to the original networks, and dashed lines to the ones with shuffled channels. K is the total depth of the networks. Experiments with additional architectures are reported in figure C4 in appendix C.

pooling in CNNs, we refer to it as *spatial* pooling to stress that the sum runs over the spatial indices of the representation. Even if there are no pooling layers at initialization, they can be realized by having homogeneous filters, i.e. all the $F \times F$ entries of $\boldsymbol{w}_{c,c'}^{k+1}$ are the same. Therefore, the closer the filters are to the homogeneous filter, the more they decrease the sensitivity of the representation to local translations.

Representations insensitive to other transformations via channel pooling. The example of translations shows that invariance can be built by first constructing an equivariant representation, and then pooling it. Invariant representations can also be built by pooling *across* channels. A two-channel example is shown figure 2, panel (b), where the filter of the second channel is built to produce the same output as the first channel when applied to a rotated input. The same idea can be applied more generally, e.g. to the other components of diffeomorphisms-such as local stretchings and shears. Below, we refer generically to any operation that builds representations invariant to diffeomorphisms by assembling distinct channels as *channel pooling*.

Disentangling spatial and channel pooling. The relative sensitivity to diffeomorphisms R_k of the kth layer representation f_k decreases after each layer, as shown in figure 3. This implies that spatial and channel pooling are carried out along the whole network. To disentangle their contribution we perform the following experiment: shuffle at random the connections between channels of successive convolutional layers, while keeping the weights unaltered. Channel shuffling amounts to randomly permuting the values of c, c' in equation (2), therefore it breaks any channel pooling while not affecting the individual filters. The values of R_k for deep networks after channel shuffling are reported in figure 3 as dashed lines and compared with the original values of R_k in solid lines. If only spatial pooling was present in the network, then the two curves would overlap. Conversely, if the decrease in R_k was all due to the interactions between channels, then the shuffled curves should be constant. Given that neither of these scenarios arises, we conclude that both kinds of pooling are being performed.

Emergence of spatial pooling after training. To bolster the evidence for the presence of spatial pooling, we analyze the filters of trained networks. Since spatial pooling can be built by having homogeneous filters, we test for its presence by looking at the frequency content of learned filters $\boldsymbol{w}_{c,c'}^{k}$. In particular, we consider the average squared projection of filters onto 'Fourier modes' $\{\Psi_l\}_{l=1,\dots,F^2}$, taken as the eigenvectors of the discrete Laplace operator on the $F \times F$ filter grid[2]. The squared projections averaged over channels read

$$\gamma_{k,l} = \frac{1}{H_{k-1}H_k} \sum_{c=1}^{H_k} \sum_{c'=1}^{H_{k-1}} \left[\Psi_l \cdot \boldsymbol{w}_{c,c'}^{k}\right]^2, \tag{3}$$

and are shown in figure 4, 1st and 2nd row. When training a deep network such as VGG11 (with and without batch-norm) [40][3] on CIFAR10, filters of layers 2–6 become low-frequency with training, while layers 1, 7, and 8 do not. Accordingly, larger gaps between dashed and solid lines in figure 3(left) open at layers 1, 7, 8:

[2] The discrete Laplacian is defined on the adjacency matrix of a graph, where each node in an $F \times F$ grid is connected to its horizontal and vertical neighbors, without considering the diagonal ones.

[3] Our main analysis focuses on VGG architectures due to their foundational significance in computer vision. Introduced in 2014 by [40], VGG models set the state-of-the-art upon their release, thanks to their straightforward yet effective design. Comprising essential

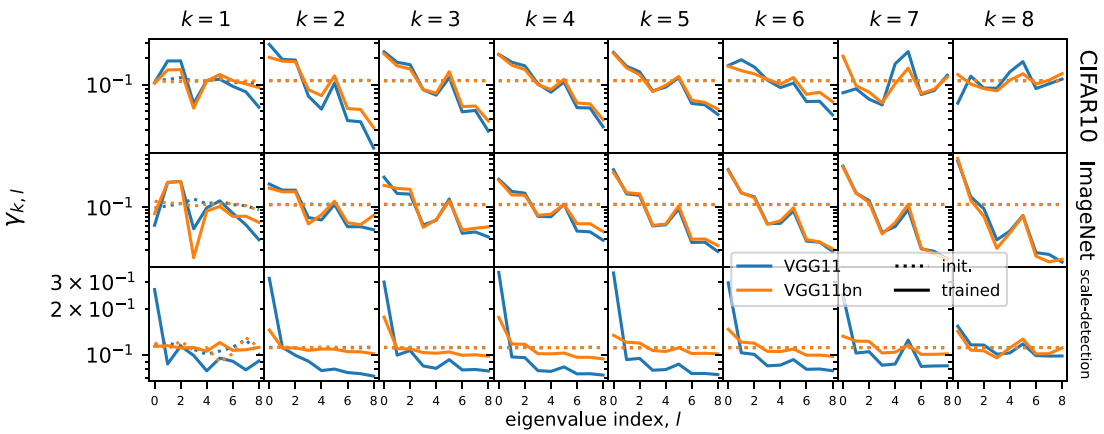

Figure 4. Projections of the network filters for VGG11 and VGG11bn onto the 9 eigenvectors of the (3×3)-grid Laplacian when training on CIFAR10 (1st row), ImageNet, (2nd row) and the scale-detection task (3rd row): dotted and solid lines correspond to initialization and trained networks, respectively. The x-axis reports low to high frequencies from left to right. Deeper layers are reported in the rightmost panels. Low-frequency modes are the dominant components in layers 2–6 when training on CIFAR10, in layers 2–8 for ImageNet. The first (constant) mode has most of the power throughout the network for scale-detection task 1. An aggregate measure of the frequency content of filters is reported in figure C5 in appendix C together with experiments with additional architectures (figures C6 and C7).

reduction in sensitivity is not due to spatial pooling in these layers. Moreover, the fact that the two dashed curves overlap is consistent with the frequency content of filters being the same for the two architectures after training. In the case of ImageNet, filters at all layers become low-frequency, except for $k = 1$.

3. Simple scale-detection tasks capture real-data observations

To sum up, the empirical evidence presented in section 2 indicates that *(i)* the generalization performance of deep CNNs correlates with their insensitivity to diffeomorphisms and sensitivity to Gaussian noise (figure 1); *(ii)* deep CNNs build their sensitivities layer by layer via spatial and channel pooling. We introduce now two idealized scale-detection tasks where the phenomena *(i)* and*(ii)* emerge again, and we can isolate the contribution of spatial pooling. Given that the structure of these tasks is simpler than real data, we can understand quantitatively how spatial pooling builds up insensitivity to diffeomorphisms and sensitivity to Gaussian noise, as we show in section 4.

Definition of scale-detection tasks. Consider input images x consisting of two active pixels on an empty background.

Task 1: Inputs are classified by comparing the euclidean distance d between the two active pixels and some *characteristic scale* ξ, as in figure 5, left. Namely, the label is $y = \text{sign}\,(\xi - d)$.

Notice that a small diffeomorphism of such images corresponds to a small displacement of the active pixels. Specifically, each of the active pixels is moved to either of its neighboring pixels or left in its original position with equal probabilities[4]. By introducing a gap g such that $d \in [\xi - g/2, \xi + g/2]$, task 1 yields an invariance to displacements of size smaller than g. Therefore, we expect that a neural network trained on task 1 will lose any information on the exact location of the active pixels within the image, thus becoming insensitive to diffeomorphisms. Intuitively, spatial pooling up to the scale ξ is the most direct mean to achieve such insensitivity. The result of the integration depends on whether none, one, or both of the active pixels lie within the pooling window, thus it is still informative of the task. We will show empirically that this is indeed the solution reached by trained CNNs.

Task 2: Inputs are partitioned into nonoverlapping patches of size ξ, as in figure 5, right. The label y is $+1$ if the active pixels fall within the same patch, -1 otherwise.

CNN components like convolutional and pooling layers, VGGs serve as ideal candidates for our study aimed at understanding the core mechanisms of CNNs. Experiments with additional architectures are reported in appendix C.

[4] We fix the length of these displacements to 1 pixel because*(i)* is the smallest value that prevents the use of pixel interpolation, which would make one active pixel an extended object *(ii)* allows for the analysis of section 4.

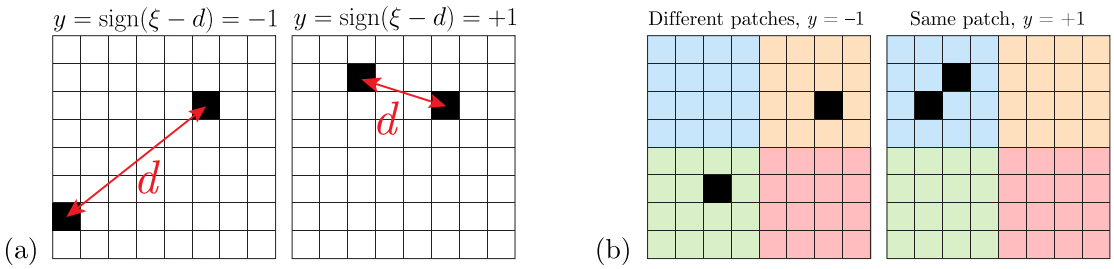

Figure 5. Example inputs for the scale-detection tasks. Task 1 (a): the label depends on whether the Euclidean distance d is larger (left) or smaller (right) than the characteristic scale ξ. Task 2 (b): the label depends on whether the active pixels belong to the same patch of size ξ (right) or not (left)—patches are shown in different colors.

In task 2, the irrelevant information is the location of the pixels within each of the non-overlapping patches. The simplest means to lose such information requires to couple spatial pooling with a stride of the size of the pooling window itself.

Same phenomenology as in real image datasets. We train the same networks used in section 2 on the scale-detection tasks. We perform SGD on the hinge loss and learning rate of 0.05. Although these scale-detection tasks are much simpler than standard benchmark datasets, deep networks trained on task 1 display the same phenomenology highlighted in section 2 for networks trained on CIFAR10 and ImageNet. First, the test error is positively correlated with the sensitivity to diffeomorphisms of the network predictor (figure C1, left panel, in appendix C) and negatively correlated with its sensitivity to Gaussian noise (middle panel) for a whole range of architectures. As a result, the error correlates well with the relative sensitivity R_f (right panel). Secondly, the internal representations of trained networks f_k become progressively insensitive to diffeomorphisms and sensitive to Gaussian noise through the layers, as shown in figure C2 in appendix C. Importantly, the curves relating sensitivities to the relative depth remain essentially unaltered if the channels of the networks are shuffled (shown as dashed lines in figure C2). We conclude that, on the one hand, channel pooling is negligible, and, on the other hand, all channels are approximately equal to the mean channel. Finally, direct inspection of the filters (figure 4, bottom row) shows that the 0-frequency component grows much larger than the others over training for layers 1–7, which are the layers where R_k decreases the most in figure C2. Filters are thus becoming nearly homogeneous, which means that the convolutional layers become effectively pooling layers.

4. Theoretical analysis of sensitivities in scale-detection tasks

We now provide a scaling analysis of the sensitivities to diffeomorphisms and noise in the internal representations of simple CNNs trained on the scale-detection tasks of section 3. It allows us to quantitatively understand how spatial pooling makes the internal representations of the network progressively more insensitive to diffeomorphisms and sensitive to Gaussian noise.

Setup. We consider simple CNNs made by stacking $\tilde{K}$ identical convolutional layers with generic filter size F, stride $s = 1$ or F and ReLU activation function $\phi(x) = \max(0, x)$. In particular, we train CNNs with stride 1 on task 1 and CNNs with stride F on task 2. For the sake of simplicity, we consider the one-dimensional version of the scale-detection tasks, but our analysis carries unaltered to the two-dimensional case. Thus, input images are sequences $x = (x_i)_{i=1,\dots,L}$ of L pixels, where $x_i = 0$ for all pixels except two. For the active pixels $x_i = \sqrt{L/2}$, so that all input images have $\|x\|^2 = L$. We will also consider single-pixel data $\delta_j = (\delta_{j,i})_{i=1,\dots,L}$. If the active pixels in x are the ith and the jth, then $x = \sqrt{L/2}\,(\delta_i + \delta_j)$. For each layer k, the internal representation $f_k(x)$ of the trained network is defined as in equation (2). The *receptive field* of the kth layer is the number of input pixels contributing to each component of $f_k(x)$. We define the *effective* receptive field A_k as the typical size of the representation of a single-pixel input, $f_k(\delta_i)$, as illustrated in red in figure 6. We denote the sensitivities of the kth layer representation with a subscript k (D_k for diffeomorphisms, G_k for noise, R_k for relative).

Assumptions. All our results are based on the assumption that the first few layers of the trained network behave effectively as a single channel with a homogeneous positive filter and no bias. The equivalence of all

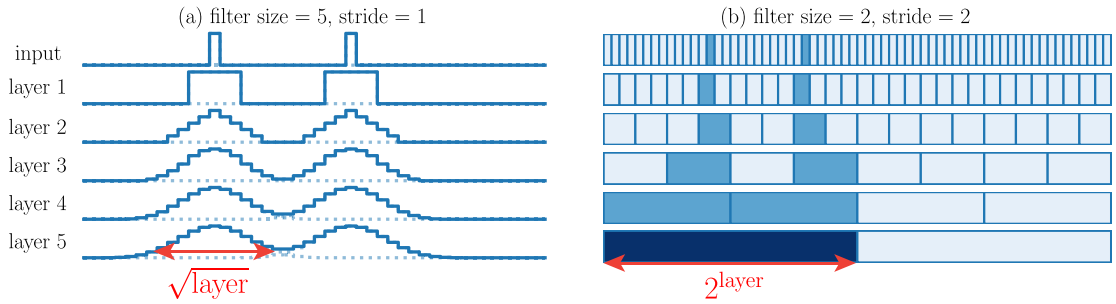

Figure 6. Hidden layers representations of simple CNNs for a scale-detection input for stride $s=1$ and filter size $F=5$ (left) and $s=F=2$ (right) when having homogeneous filters at every layer. The effective receptive field size of the last layer in the two different cases is shown in red. (Left) Every active pixel in the input becomes a Gaussian profile whose width increases throughout the network. (Right) Every neuron in layer k has activity equal to the number of active pixels that are present in its receptive field of width 2^k. The dark blue in the last layer indicates that there are two active pixels in its receptive field, while the lighter blue of the precedent layers indicates that there is just 1.

the channels with their mean is supported by figure C2 in appendix C, which shows how shuffling channels does not affect the internal representations of VGGs. In addition, figure 4(bottom row) shows that the mean filters of the first few layers are nearly homogeneous. We set the homogeneous value of each filter to keep the norm of representations constant over layers. Moreover, we implement a deformation of the input x of our scale-detection tasks as a random displacement of each active pixel at either left or right with probability 1/2.

4.1. Task 1, stride 1

For a CNN with stride 1, under the homogeneous filter assumption, the size of the effective receptive field A_k grows as $\sqrt{k}$. A detailed proof is presented in the appendix A and figure 6, left panel, shows an illustration of the process. Intuitively, applying a homogeneous filter to a representation is equivalent to making each pixel diffuse, i.e. distributing its intensity uniformly over a neighborhood of size F. With a single-pixel input δ_i, the effective receptive field of the kth layer $f_k(\delta_i)$ is equivalent to a k-step diffusion of the pixel, thus it approaches a Gaussian distribution of standard deviation $\sqrt{k}$ centered at i. The size A_k is the standard deviation, thus $A_k \sim \sqrt{k}$. In other words, using the language of digital image processing [41], applying k moving averages is equivalent to applying a Gaussian filter with standard deviation proportional to $\sqrt{k}$, for large enough k. The proof we present in appendix A requires large depth $\tilde{K} \gg 1$ and large image width $L \gg F\tilde{K}^{1/2}$ and the empirical studies of section 3 satisfy these constraints ($F \sim 3$, $L \sim 32$ and $\tilde{K} \sim 10$).

We remark that at initialization, $f_k(x)$ behave, in the limit of large number of channels and width (and small bias), as Gaussian random fields with correlation matrix $\mathbb{E}[f_k(x)f_k(y)] \approx \delta(x-y)$, with δ the Dirac delta [42, 43]. This spiky correlation matrix implies that for any perturbation $y = x + \varepsilon$, the representation $f_k(y)$ changes with respect to $f_k(x)$ independently on ε. This behavior is remarkably different to the smooth case achieved by the diffusion, after training. Consequently, both D_k and G_k are constant with respect to k at initialization. This is consistent with the observations reported in figure 7.

Sensitivity to diffeomorphisms. Let i and j denote the active pixels locations, so that $x \propto \delta_i + \delta_j$. Since both the elements of the inputs and those of the filters are non-negative, the presence of ReLU nonlinearities is irrelevant and the first few hidden layers are effectively linear. Hence the representations are linear in the input, so that $f_k(x) = f_k(\delta_i + \delta_j) = f_k(\delta_i) + f_k(\delta_j)$. In addition, since the effect of a diffeomorphism is just a 1-pixel translation of the representation irrespective of the original positions of the pixels, the normalized sensitivity D_k can be approximated as follows

$$D_k \sim \frac{\|f_k(\delta_{i+1}) - f_k(\delta_i)\|_2^2}{\|f_k(\delta_i)\|_2^2}. \tag{4}$$

The denominator in equation (4) is the squared norm of a Gaussian distribution of width $\sqrt{k}$, $\|f_k(v_i)\|_2^2 \sim k^{-1/2}$. The numerator compares f_k with a small translation of itself, thus it can be approximated by the squared norm of the derivative of the Gaussian distribution, $\|f_k(\delta_{i+1}) - f_k(\delta_i)\|_2^2 \sim k^{-3/2}$. Consequently, we have

$$D_k \sim k^{-1} \sim A_k^{-2}. \tag{5}$$

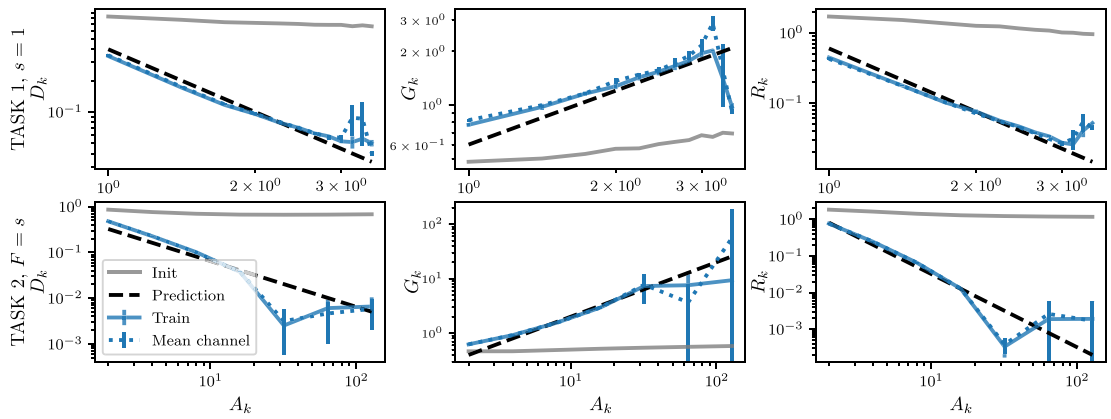

Figure 7. Sensitivities of internal representations f_k of simple CNNs against the kth layer receptive field size A_k for trained networks (solid blue) and at initialization (solid gray). The top row refers to task 1 with $s = 1$ and $F = 3$; the bottom row to task 2 with $F = s = 2$. For the first large part of the network, the sensitivities obtained by replacing each layer with the mean channel (blue dotted) overlap with the original sensitivities. Predictions equations (5) and (7) for task 1 and equations (8) and (9) for task 2 are shown as black dashed lines.

Note that this prediction does not depend on the activation function, as backed by the experimental results in appendix B using the tanh activation function.

Sensitivity to Gaussian noise. To analyze G_k one must take into account the rectifying action of ReLU, which sets all the negative elements of its input to zero. The first ReLU is applied after the first homogeneous filters, thus the zero-mean noise is superimposed on a patch of F active pixels. Outside such a patch, only positive noise terms survive. Within the patch, being summed to a positive background, negative terms can survive the rectification of ReLU. Nevertheless, if the size of the image is much larger than the filter size, the contribution from active pixels to G_k is negligible and we can approximate the difference between noisy and original representations $f_1(x+\eta) - f_1(x)$ with the rectified noise $\phi(\eta)$. After the first layer, the representations consist of non-negative numbers, thus we can forget again the ReLU and write

$$G_k \sim \frac{\mathbb{E}_\eta \| f_k(\phi(\eta)) \|_2^2}{\| f_k(\delta_i) \|_2^2}. \tag{6}$$

Repeated applications of homogeneous filters to the rectified noise $\phi(\eta)$ result again in a diffusion of the signal. Since $\phi(\eta)$ has different independent and identically distributed non-zero entries for different realizations of η, averaging over η is equivalent to considering a homogeneous profile for $f_k(\phi(\eta))$. As a result, the numerator in equation (6) is a constant independent of k. The denominator is the same as in equation (4), $\|f_k(\delta_i)\|_2^2 \sim k^{-1/2}$, hence

$$G_k \sim k^{1/2} \sim A_k, \tag{7}$$

i.e. the sensitivity to Gaussian noise grows as the size of the effective receptive fields. From the ratio of equations (5) and (7), we get $R_k \sim A_k^{-3}$.

Without rectification, the entries of $f_k(\eta)$ are given by a sum of Gaussian numbers η_i within the effective receptive field $A_k \sim \sqrt{k}$, rescaled by a factor $1/\sqrt{k}$ [5]. Such sum, for the Central Limit Theorem, scales as $k^{-1/4}$ and consequently the numerator in equation (6) scales as $k^{-1/2}$, yielding a constant G_k. We conclude that the rectifying action of ReLU is crucial in building up sensitivity to noise. For an odd activation function like tanh a spatial pooling solution is not sensitive to noise, as supported by our experiments in appendix B. We expect that a given activation function makes the network less sensitive to noise according to how much is close to being odd. For example, the swish$(x) = x/(1+e^{-\beta x})$ activation function, which is linear for $\beta = 0$ and a ReLU for $\beta = \infty$, has a region Z_β around the origin where is approximately linear, whose width is controlled by β. Given a value of β, the network will not sense noise of magnitude smaller than Z_β, while it will be affected by noise with larger magnitude.

[5] This is due to the aforementioned fact that $f_k(\delta_i)$ approaches a Gaussian with standard deviation $\sqrt{k}$ centered in i. The factor $1/\sqrt{k}$ is the normalization of such Gaussian.

4.2. Task 2, stride equal filter size

When the stride s equals to the filter size F the number of pixels of the internal representations is reduced by a factor F at each layer, thus f_k consists of L/F^k pixels. Meanwhile, the effective size of the receptive fields grows exponentially at the same rate: $A_k = F^k$ (see figure 6, left for an illustration).

Sensitivity to diffeomorphisms. For a given layer k, consider a partition of the input image into L/F^k patches. Each pixel of f_k only looks at one such patch and its intensity coincides with the number of active pixels within the patch. As a result, the only diffeomorphisms that change f_k are those which move one of the active pixels from one patch to another. Since active pixels move by 1, this can only occur if one of the active pixels was originally located at the border of a patch, which in turn occurs with probability $\sim 1/F^k$. In addition, the norm $\|f_k(\delta_i)\|_2$ at the denominator does not scale with k, so that

$$D_k \sim F^{-k} \sim A_k^{-1}. \tag{8}$$

Sensitivity to Gaussian noise. Each pixel of f_k looks at a patch of the input of size F^k, thus f_k is affected by the sum of all the noises acting on such patch. Since these noises have been rectified by ReLU, by the central limit theorem the sum scales as the number of summands F_k. Thus, the contribution of each pixel of f_k to the numerator of G_k scales as $(F^k)^2$. As there are L/F^k pixels in f_k, one has

$$G_k \sim \left(F^k\right)^2 \left(L/F^k\right) \sim F^k \sim A_k. \tag{9}$$

Without rectification, the sum of F^k independent noises would scale as the square root of the number of summands F^k, yielding a constant G_k. The scaling $R_k \sim A_k^{-2}$ follows from the ratio of equations (8) and (9).

4.3. Comparing predictions with experiments

We present the experimental setup used to train simple CNNs introduced in section 4, to test the predictions for the sensitivities equations (5) and (7) for task 1 and equations (8) and (9) for task 2. For task 1, we use the scale-detection task in the version of figure 5(b), with $\xi = 11$ and gap $g = 4$ and image size $L = 32$. For this task, we use CNNs with stride $s = 1$ and filter size $F = 3$. The width of the CNN is fixed to 1000 channels since we observe that a higher number of channels does not affect the training dynamics. The depth K has to be such that the effective receptive field A_K is wide enough to solve the task, recognizing the scale $\xi - g$.

For the training, we use $P = 48$ training points and SGD with learning rate of 0.01 and batch size 8. We use weight decay for the L_2 norm of the filter weights with ridge 0.01. These hyper-parameters are found by an extensive search and they allow to solve the task in a reasonable computing time. We stop the training after 500 times the interpolation time, which is the time required by the network to reach zero interpolation error of the training set. The goal of this extended training time is to reach the solution with minimal norm thanks to weight decay. The generalization error with that number of training points P of the trained CNNs is exactly zero: they learn spatial pooling perfectly.

We show the sensitivities of the trained CNNs, averaged over 4 seeds, in the top panels of figure 7, where we also successfully test the predictions equations (5) and (7). We remark that to compute G_k we inserted Gaussian noise with already the ReLU applied on since we observe that without it we would see a pre-asymptotic behavior for G_k with respect to A_k.

For the dataset in task 2, we use the block-wise version of the Scale-Detection task shown in figure 5(c), fixing $\xi = 2^5$ and $L = 2^7$. We use CNNs with structure tuned to this task, with stride equal to filter size $s = F = 2$. We use 7 layers and 1000 channels for the CNNs, fixed with the same logic as in task 1. The training is performed using SGD and weight decay with the same parameters as in task 1. We use $P = 2^{10}$ training points since it allows reaching zero test error. In the bottom panels of figure 7 we show that the predictions equations (8) and (9) capture the experimental results, averaged over 10 seeds.

Notice that if all the filters at a given layer are replaced with their average, the behavior of the sensitivities as a function of depth does not change (compare solid and dotted blue curves in the figure). This confirms our assumption that all channels behave like the mean channel. In addition, tables C2 and C3 in appendix C show that the mean filters are approximately homogeneous. This is consistent with the spectral analysis of the filters shown in the top row in figure B2.

5. Conclusion

The meaning of an image often depends on local features, as evidenced by the fact that artists only need a small number of strokes to represent a visual scene. Since the exact locations of the features are not important in determining the image class, diffeomorphisms of limited magnitude leave the class unchanged. Indeed, as shown in [15], (former) state-of-the-art deep convolutional networks learn the insensitivity to

diffeomorphisms during training. Here, we have shown how such insensitivity is built via learning a combination of spatial and channel pooling. Interestingly, in ReLU networks, spatial pooling comes together with an increased sensitivity to random noise in the image, as captured and quantified in our simple scale-detection model of data. This phenomenon does not appear with an odd activation function like the tanh, as shown in appendix B. Our analysis suggests that to build trained neural networks robust to random noise with a given magnitude, it may be useful to use activation functions odd-shaped in a region with a width larger than such magnitude. How this would affect the performance is left for future work.

It is commonly believed that the best architectures are those that extract the data features that are the most relevant for the task. The pooling operations studied here, which allow the network to forget the exact locations of these features, are probably more effective when features are better extracted. This point may be responsible for the observed strong correlations between the network performance and its stability to diffeomorphisms. Designing synthetic models of data whose features are combinatorial and stable to smooth transformations is very much needed to clarify this relationship, and ultimately understand how deep networks learn high-dimensional tasks with limited data.

Data availability statement

The data that support the findings of this study are openly available at the following URL/DOI: https://github.com/leonardopetrini/relativestability/blob/main/experiments.md.

Acknowledgments

We thank Alessandro Favero, Antonio Sclocchi for helpful discussions. This work was supported by a grant from the Simons Foundation (#454953 Matthieu Wyart).

Appendix A. Task 1, stride 1: proofs

In section 4.1 we consider a simple CNN with stride $s=1$ and filter size F trained on scale-detection task 1. We fix the total depth of these networks to be $\tilde{K}$. We postulated in section 4 that this network displays a one-channel solution with homogeneous filter $[1/F,\dots,1/F]$ and no bias. We can understand the representation $f_k(x)$ at layer k of an input datum x by using single-pixel inputs δ_i. Let us recall that these inputs have all components to 0 except the ith, set to 1. Then, we have that a general datum x is given by $x \propto (\delta_i + \delta_j)$, where i and j are the locations of the active pixel in x. We have argued in the main text that the representation $f_k(\delta_i)$ is a Gaussian distribution with width $\sqrt{k}$. In this appendix, we prove this statement.

First, we observe that in this solution, since both the elements of the filters and those of the inputs are non-negative, the network behaves effectively as a linear operator. In particular, each layer corresponds to the application of a $L \times L$ circulant matrix M, which is obtained by stacking all the L shifts of the following row vector,

$$\left[\underbrace{1,1,\dots,1}_{F}\underbrace{0,0,0,\dots,0}_{L-F}\right]. \tag{A.1}$$

with periodic boundary conditions. The first row of such a matrix is fixed as follows. If F is odd the patch of size F is centered on the first entry of the first row, while if F is even we choose to have $(F/2)$ ones at the left of the first entry and $(F/2)-1$ at its right. The output f_k of the layer k is then the following: $f_k(\delta_i) = M^k\delta_i$.

Proposition 1. *Let's consider the $L \times L$ matrix M and a given L vector δ_i, as defined above. For odd $F \geqslant 3$, in the limit of large depth $\tilde{K} \gg 1$ and large width $\tilde{L} \gg F\sqrt{\tilde{K}}$, we have that*

$$\begin{aligned}\left(M^k\right)_{ab}\delta_i &= \frac{1}{2\sqrt{\pi}\sqrt{D^{(1)}}\sqrt{k}}e^{-\frac{(a-i)^2}{4D^{(1)}k}},\\ D^{(1)} &= \frac{1}{12F}(F-1)^3,\end{aligned} \tag{A.2}$$

while for even F:

$$\left(M^k\right)_{ab}\delta_i = \frac{1}{2\sqrt{\pi}\sqrt{D^{(2)}}\sqrt{k}}e^{-\frac{\left(v^{(2)}k+a-i\right)^2}{4D_F^{(2)}k}}, \qquad D^{(2)} = \frac{1}{12F}\left(F^3-3F^2+6F-4\right), \tag{A.3}$$

with $v^{(2)} = (1-F)/(2F)$.

Proof. The matrix M can be seen as the stochastic matrix of a Markov process, where at each step the random walker has uniform probability $1/F$ to move in a patch of width F around itself. We write the following recursion relation for odd F,

$$p_{a,i}^{(k+1)} = \frac{1}{F}\left(p_{a-(F-1)/2,i}^{(k)}+\cdots+p_{a,i}^{(k)}+\cdots+p_{a+(F-1)/2,i}^{(k)}\right), \tag{A.4}$$

and even F,

$$p_{a,i}^{(k+1)} = \frac{1}{F}\left(p_{a-F/2,i}^{(k)}+\cdots+p_{a,i}^{(k)}+\cdots+p_{a+(F/2-1),i}^{(k)}\right). \tag{A.5}$$

In any of these two cases, this is the so-called master equation of the random walk (see [44] in section 1.4, page 11 for a generalized version of the master equation). In the limit of large image width L and large depth $\tilde{K}$, we can write the related equation for the continuous process $p_i(a,k)$, which is called Fokker–Planck equation in physics and chemistry [44] or forward Kolmogorov equation in mathematics [45],

$$\partial_k p_{a,i}^{(k)} = v\partial_a p_{a,i}^{(k)} + D\partial_a^2 p_{a,i}^{(k)}. \tag{A.6}$$

where the drift coefficient v and the diffusion coefficient D are defined in terms of the probability distribution $W_i(x)$ of having a jump x starting from the location i

$$v = \int \mathrm{d}x W_i(x)\,x, \qquad D = \int \mathrm{d}x W_i(x)\,x^2. \tag{A.7}$$

In our case we have $W_i(x) = 1/F$ for $x \in [i-(F-1)/2, i+(F-1)/2]$ for odd F and $x \in [i-F/2, i+F/2-1]$ for even F, yielding the solutions for the Fokker–Planck equations for even and odd F reported in equations (A.2) and (A.3).

We can better characterize the limits of large image width L and large network depth $\tilde{K}$ as follows. The proof relies on the fact that a random walk, after a large number of steps, converges to a diffusion process. Here the number of steps is given by the depth $\tilde{K}$ of the network. Consequently, we need $\tilde{K} \gg 1$. Moreover, we want that the diffusion process is not influenced by the boundaries of the image, of width L. The average path walked by the random walker after $\tilde{K}$ steps is given by $F\sqrt{\tilde{K}}$. Then, we require $F\sqrt{\tilde{K}} \ll L$.

□

Appendix B. Odd activation function

In this section, we investigate the learned solution and the sensitivities of vanilla CNNs with an odd activation function such as tanh, trained on the first scale-detection task shown in figure 5, and how they differ with respect to the ones of vanilla ReLU CNN presented in section 4.

Experimental setup The network setup is the same as in section 4 for task 1: we consider vanilla CNNs made by stacking up convolutional layers with filter size F and stride $s = 1$, with tanh activation function. The experimental scheme for the data and the network is the same as in section 4.3 for task 1. The differences in the training scheme are the following: (i) we consider a learning rate of 0.003 instead of 0.01 since it yields a more stable training and (ii) we train up to 200 times the interpolation time instead of 500 times since the interpolation time is larger of a factor 10 with respect to ReLU CNNs.

Frequency content of filters We first look at the kind of solution learned by the CNN. As done in section 2, we analyze the spatial frequency content of the learned filters of each layer of the CNN, to see whether spatial pooling has been learned with training. We remind that a spatial pooling solution would yield fully homogeneous filters. In particular, as in section 2, to perform the spectral analysis of the filters $\boldsymbol{w}_{c,c'}^{k}$, where k is the index layer and c and c' are the indices of the H_{k-1} and H_k channels of the layer $k-1$ and

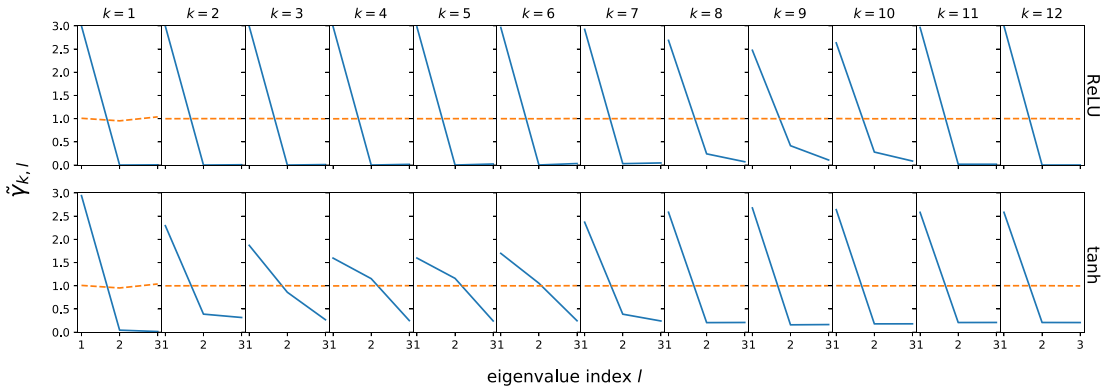

Figure B1. Projections $\tilde{\gamma}_{k,l}$ (defined in equation (B.1)) of vanilla ReLU (top row) and tanh (bottom row) CNN filters onto the 3 eigenvectors of the 3-grid Laplacian when training on the first scale-detection task presented in figure 5(a), before (dashed lines) and after training (solid lines). Each plot is for a different layer with index k, going deeper in the network from left to right. For each plot, eigenvector indices l are reported on the x-axis for increasing frequencies from left to right. The filters of each layer for both ReLU and tanh CNNs acquire a dominant low-frequency component, which is less significant for layers 3–6 in the tanh CNNs.

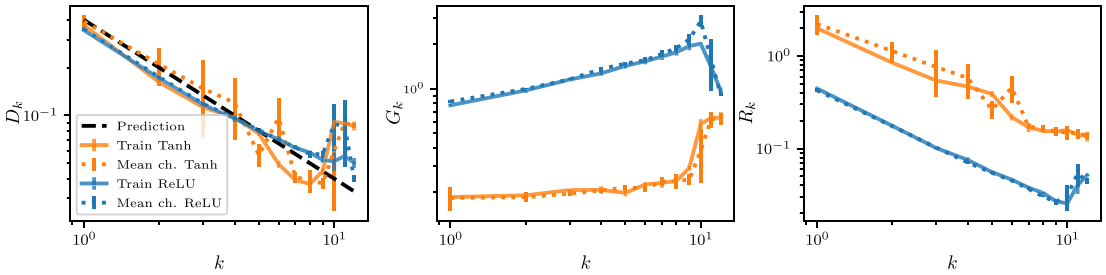

Figure B2. Sensitivities of internal representations f_k of trained vanilla CNNs against the kth layer, using the ReLU activation function (blue solid lines) and tanh activation function (orange solid lines). The dotted lines refer to the sensitivities of the same networks where each layer is replaced by its mean channel (blue line for ReLU and orange line for tanh). The prediction equation (5) of D_k (black dashed line) holds for both tanh and ReLU for the first part of the network. The behavior of G_k of the trained tanh CNN is instead very close to flat in the first large part of the network, consistent with the analysis in section 4.1.

k, we look at the projections of the filters onto the eigenvectors $\{\Psi_l\}_{l=1,\dots,F}$ of the discrete Laplace operator defined on the $F-$filter grid:

$$\tilde{\gamma}_{k,l} = \langle \left[\Psi_l \cdot \boldsymbol{w}^k_{c,c'}\right]^2 \rangle_{H_{k-1},H_k} \big/ \langle ||\boldsymbol{w}^k_{c,c'}||_2^2 \rangle_{H_{k-1},H_k}, \tag{B.1}$$

which is essentially the same as the one considered in the main text in equation (3), renormalized by the layer-wise norm of the filters, averaged over the channel indices[6]. Since lower l stands for lower frequency, the layer k will be closer to spatial pooling as $\tilde{\gamma}_{k,l}$ is larger for small l with respect to large l. We observe that ReLU networks become remarkably low-frequency throughout all the network, as shown in the top row in figure B1. Tanh CNNs, while learning solutions predominantly low-frequency, show instead for the intermediate layers 3–6 a higher projection onto high-frequency components, in the bottom row in figure B1. This suggests that the tanh CNN solution is dominated by spatial pooling, but not completely.

Sensitivities We look at the sensitivities of trained tanh and ReLU CNNs with respect to the index layer k in figure B2. We observe that D_k of trained ReLU and tanh CNNs, in the left panel, are similar in the first part of the network, consistently with the theoretical analysis in section 4.1 which is independent of the activation function and it is solely based on the assumption of a spatial pooling solution. Such assumption is supported empirically for ReLU networks by figure B1, whilst for tanh networks we observe that few layers learn a significant projection also onto high-frequency components, even though the dominant projection is still the lowest frequency one. This slight departure from a full spatial pooling solution has two consequences: (i) the trained tanh D_k departs from the theoretical prediction after the very first few layers (ii) the sensitivity of the tanh CNN where each layer is replaced by its mean channel shows a gap with the trained CNN sensitivity.

[6] This renormalization is to cope with the vanishing gradient issue.

The sensitivity G_k to white noise in the middle panel in figure B2 is significantly lower for the tanh with respect to the ReLU network. Furthermore, the tanh sensitivity is almost constant for a large part of the network. This observation supports the theoretical analysis carried out in section 4. According to that analysis, the sensitivity to noise in ReLU networks increases due to the perturbing noise that, after being rectified by the ReLU units, interferes constructively and piles up more and more throughout the network. For odd activation functions like the tanh, both positive and negative noises pass through them, interfering destructively and not affecting the network representations. Consequently, for such activation functions, trained networks with a spatial pooling solution are not sensitive to noise. This is supported by the middle panel in figure B2. Since the behavior of the tanh G_k is significantly sub-dominant with respect to the ReLU G_k, while the D_k for both networks are similar, the tanh R_k decreases less with k than the ReLU R_k, as shown in the right panel in figure B2.

Appendix C. Additional figures and tables

Table C1 provides a comprehensive summary of the main characteristics of the various neural network architectures studied in this paper. It details essential features such as the depth of the network, the number of parameters, and the activation functions used, among other attributes. This table serves as a quick reference for understanding the configurations of each architecture.

To support the assumption done in section 4 that the trained CNNs are effectively behaving as one channel with homogeneous positive filters, we report the numerical values of the average filter over channels per layer in table C2 for task 1 and table C3 for task 2. They are positive in the first nine hidden layers, where channel pooling is most pronounced.

Table C1. Network architectures and their main characteristics Summary of network architectures featured in this study, along with key properties.

Features	FullConn	LeNet [46]	AlexNet [47]
Depth	2, 4, 6	5	8
Num. Parameters	200k	62k	23 M
FC layers	2, 4, 6	3	3
Activation	ReLU	ReLU	ReLU
Pooling	—	Max	Max
Dropout	—	—	Yes
Batch norm	—	—	—
Skip connections	—	—	—
Features	VGG [40]	ResNet [48]	EfficientNetB0 [49]
Depth	11, 16, 19	18, 34, 50	18
Num. parameters	9–20 M	11–24 M	5
FC layers	1	1	1
Activation	ReLU	ReLU	Swish
Pooling	Max	Avg. (last)	Avg. (last)
Dropout	—	—	Yes + Dropconnect
Batch norm	Conditional	Yes	Yes
Skip connections	—	Yes	Yes (inv. residuals)

Table C2. Average over channels of filters in layer k, before and after training, for simple CNNs with $s = 1$ and $F = 3$ trained on task 1. The network learns filters that are much more homogeneous than initialization.

	Init.	After training
$k = 1$	$[0.0132, 0.0023, -0.0068]$	$[0.2928, 0.2605, 0.2928]$
$k = 2$	$[0.0014, -0.0007, -0.0009]$	$[0.0039, 0.0035, 0.0039]$
$k = 3$	$[-0.0006, -0.0001, 0.0010]$	$[0.0043, 0.0038, 0.0043]$
$k = 4$	$[3.4610 \times 10^{-5}, 6.5687 \times 10^{-4}, -9.1634 \times 10^{-4}]$	$[0.0039, 0.0033, 0.0038]$
$k = 5$	$[-0.0006, 0.0002, -0.0009]$	$[0.0038, 0.0032, 0.0038]$
$k = 6$	$[0.0012, -0.0011, -0.0003]$	$[0.0038, 0.0031, 0.0038]$
$k = 7$	$[-0.0006, 0.0004, 0.0003]$	$[0.0041, 0.0032, 0.0040]$
$k = 8$	$[0.0005, -0.0012, 0.0010]$	$[0.0036, 0.0024, 0.0035]$
$k = 9$	$[0.0005, -0.0012, 0.0010]$	$[0.0021, 0.0016, 0.0017]$
$k = 10$	$[-0.0025, 0.0015, -0.0006]$	$[-0.0013, -0.0008, -0.0010]$
$k = 11$	$[-0.0006, 0.0005, 0.0009]$	$0.0002, 0.0002, 0.0002]$
$k = 12$	$[3.3418 \times 10^{-4}, 3.3521 \times 10^{-5}, 1.3936 \times 10^{-3}]$	$[0.0009, 0.0008, 0.0009]$

Table C3. Average over channels of filters in layer k, before and after training, for simple CNNs with $s = F = 2$ trained on task 2. The network learns filters that are much more homogeneous than initialization.

	Init.	After training
$k = 1$	$[-0.0559, -0.0291]$	$[0.3828, 0.3737]$
$k = 2$	$[-0.0022, 0.0010]$	$[0.0060, 0.0059]$
$k = 3$	$[0.0006, -0.0010]$	$[0.0064, 0.0065]$
$k = 4$	$[-0.0020, 0.0009]$	$[0.0059, 0.0060]$
$k = 5$	$[0.0002, 0.0008]$	$[9.9935 \times 10^{-5}, 2.1380 \times 10^{-4}]$
$k = 6$	$[-0.0003, -0.0010]$	$[-0.0028, -0.0029]$
$k = 7$	$[-7.4610 \times 10^{-4}, 8.4595 \times 10^{-5}]$	$[-0.0009, -0.0009]$

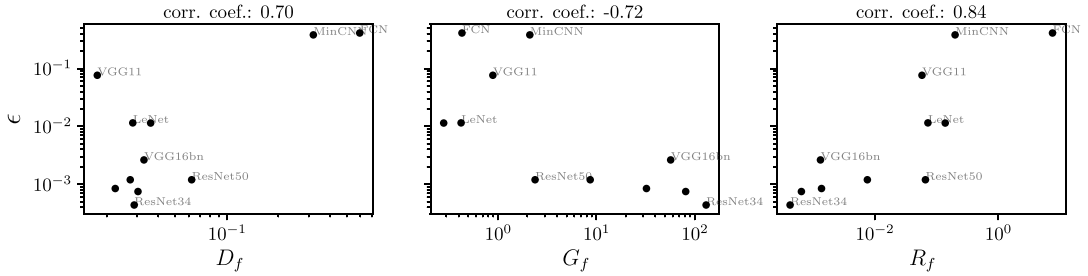

Figure C1. Generalization error ε versus sensitivity to diffeomorphisms D_f (left), noise G_f (center) and relative sensitivity R_f (right) for a wide range of architectures trained on scale-detection task 1 (train set size: 1024, image size: 32, $\xi = 14, g = 2$). As in real data, ε is positively correlated with D_f and negatively correlated with G_f. The correlation is the strongest for the relative measure R_f. CNNs are trained in the setting described in section 2, while the FCN with the ADAM optimizer, a learning rate of 0.1 and no scheduler.

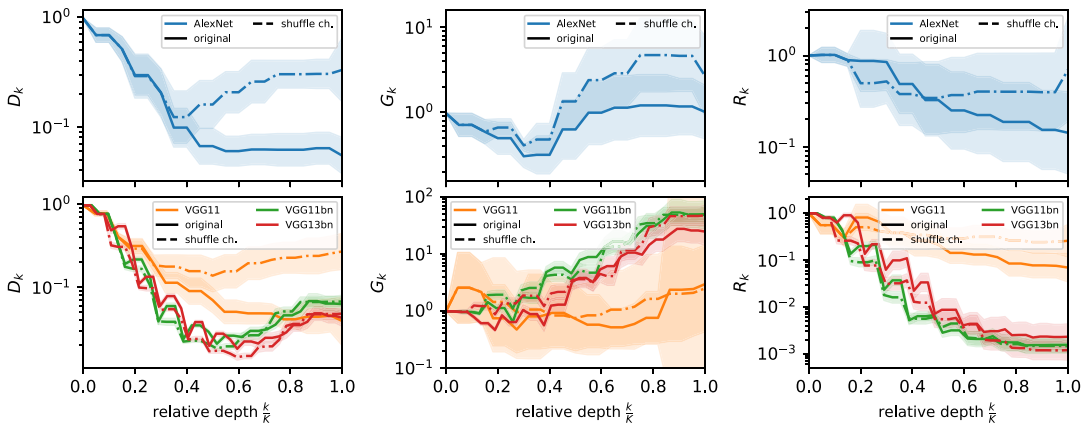

Figure C2. Sensitivities (D_k left, G_k middle and R_k right) of the internal representations vs relative depth for AlexNet (1st row) and VGG networks (2nd row) trained on scale-detection task 1. Dot-dashed lines show the sensitivities of networks with shuffled channels.

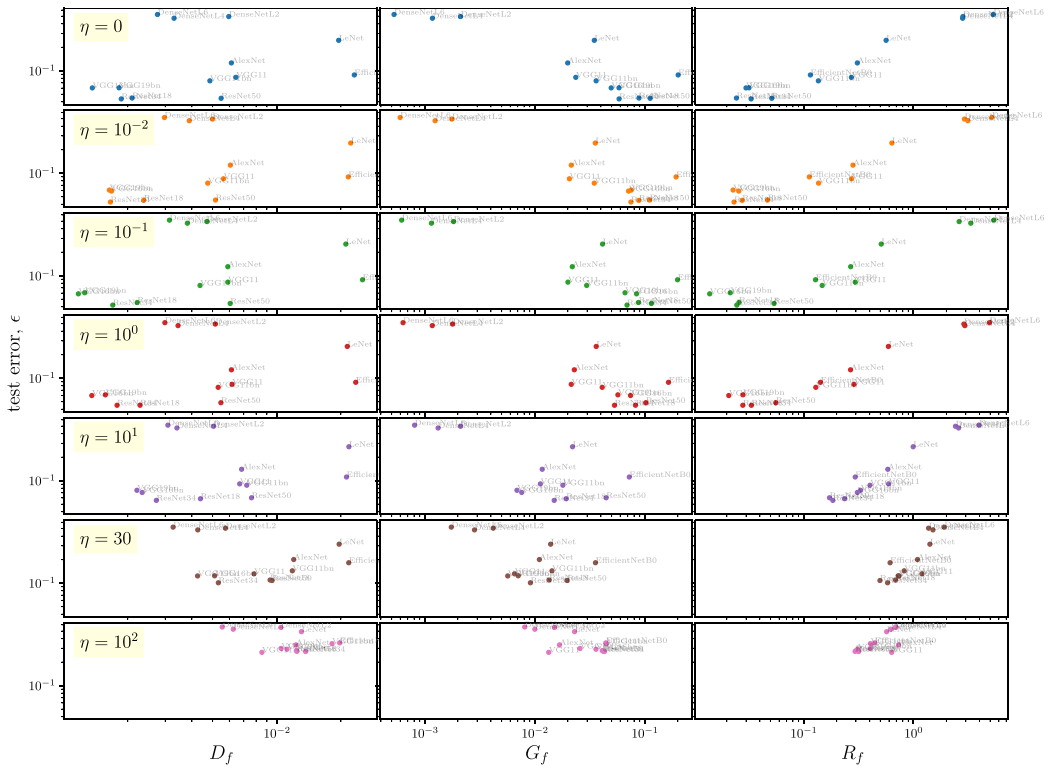

Figure C3. Test error vs. sensitivities (columns) when training on noisy CIFAR10. The different rows correspond to increasing noise magnitude η. Different points correspond to network architectures, see gray labels. The content of this figure* is also represented in compact form in figure 1, right.

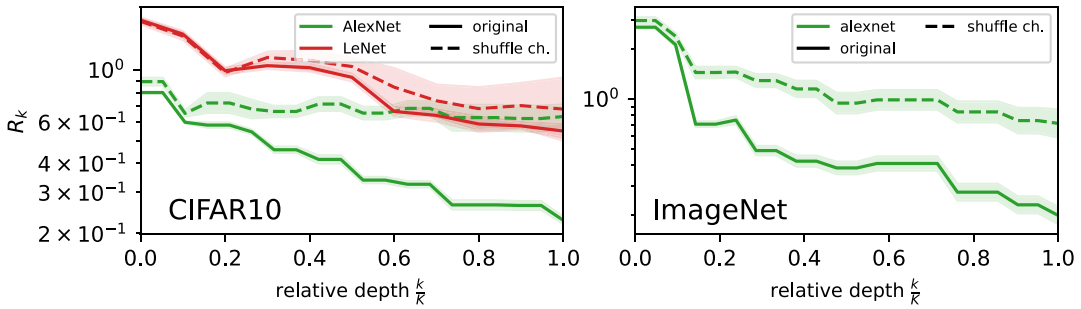

Figure C4. Analogous of figure 3 for different network architectures: relative sensitivity R_k as a function of depth for LeNet and AlexNet architectures trained on CIFAR10 (left) and ImageNet (right). Full lines indicate experiments done on the original networks, dashed lines the ones after shuffling channels. K indicates the network's total depth.

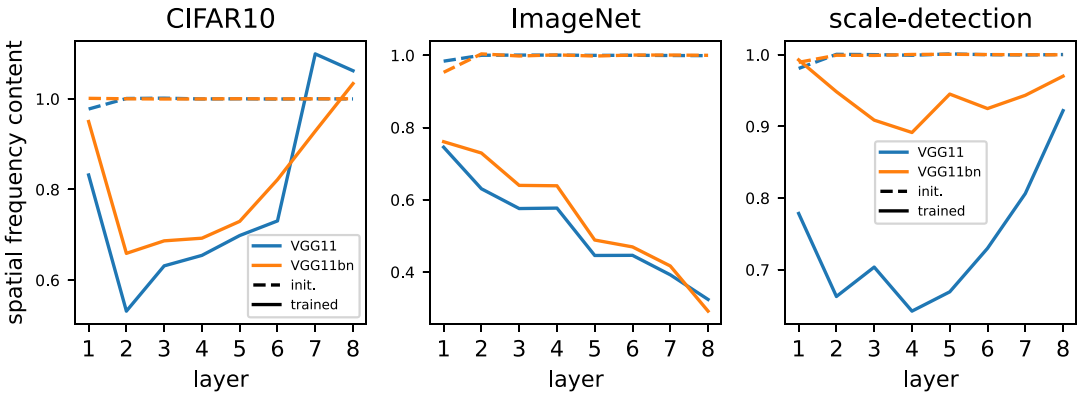

Figure C5. Spatial frequency content of filters for CIFAR10 (left), ImageNet (center), and the scale-detection task (right). The y-axis reports an aggregate measure among spatial frequencies: $N(\sum_{i=1}^{N}\lambda_l)^{-1}\langle\|\boldsymbol{w}_c^k\|^2\rangle_c^{-1}\sum_{l=1}^{F^2}\lambda_l\langle(\boldsymbol{\Psi}_l\cdot\boldsymbol{w}_c^k)^2\rangle_c$, where $\boldsymbol{\Psi}_l$ are the 3×3 Laplacian eigenvectors and λ_l the corresponding eigenvalues, $\boldsymbol{w}_c^k$ the cth filter of layer k and $\langle\cdot\rangle_c$ denotes the average over c. This is an aggregate measure over frequencies, the frequency distribution is reported in the main text, figure 4.

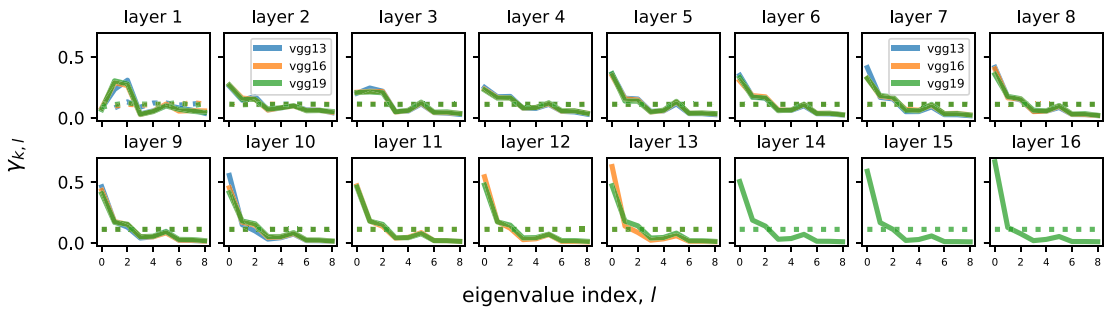

Figure C6. Analogous of figure 4 for deep VGGs trained on ImageNet. Dotted and full lines respectively correspond to initialization and trained networks. The x-axis reports low to high frequencies from left to right. Deeper layers are reported in the rightmost panels.

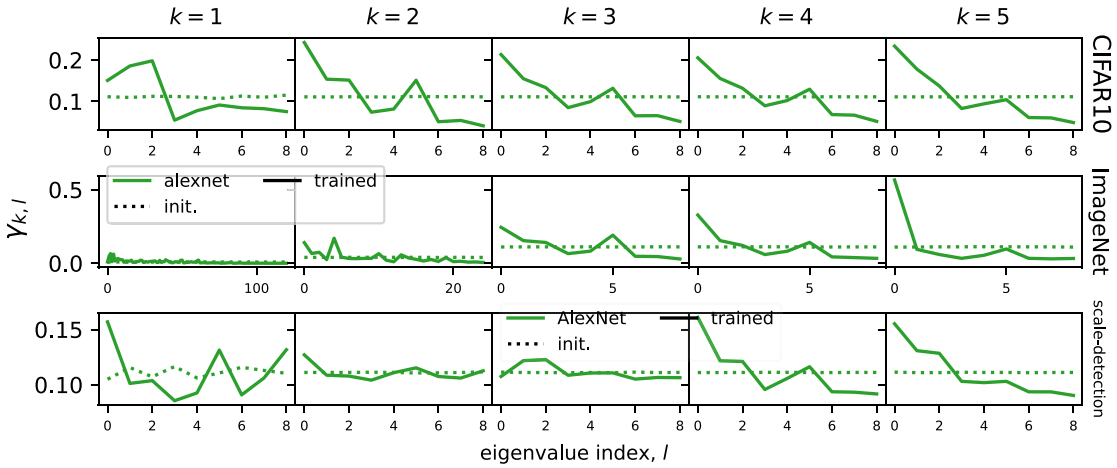

Figure C7. Analogous of figure 4 for AlexNet trained on CIFAR10 (1st row), ImageNet (2nd row) and the scale detection task (3rd row). Dotted and full lines respectively correspond to initialization and trained networks. The x-axis reports low to high frequencies from left to right. Deeper layers are reported in the rightmost panels.

ORCID iDs

Umberto M Tomasini https://orcid.org/0009-0001-0249-6850
Leonardo Petrini https://orcid.org/0000-0003-1517-679X

References

[1] Amodei D *et al* 2016 Deep speech 2: end-to-end speech recognition in english and mandarin *Proc. 33rd Int. Conf. on Machine Learning* (*Proc. Machine Learning Research* vol 48) ed M F Balcan and K Q Weinberger (PMLR) pp 173–82
[2] Huval B *et al* 2015 An empirical evaluation of deep learning on highway driving (arXiv:1504.01716)
[3] Mnih V *et al* 2015 Human-level control through deep reinforcement learning *Nature* **518** 529–33
[4] Shi B, Bai X and Yao C 2017 An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition *IEEE Trans. Pattern Anal. Mach. Intell.* **39** 2298–304
[5] Silver D *et al* 2017 Mastering the game of go without human knowledge *Nature* **550** 354–9
[6] Luxburg U v and Bousquet O 2004 Distance–based classification with Lipschitz functions *J. Mach. Learn. Res.* **5** 669–95
[7] Deng J, Dong W, Socher R, Li L, Kai Li and Li F-F 2009 ImageNet: a large-scale hierarchical image database *2009 IEEE Conf. on Computer Vision and Pattern Recognition* pp 248–55
[8] Hestness J, Narang S, Ardalani N, Diamos G, Jun H, Kianinejad H, Patwary M, Ali M, Yang Y and Zhou Y 2017 Deep learning scaling is predictable, empirically (arXiv:1712.00409)
[9] Spigler S, Geiger M and Wyart M 2020 Asymptotic learning curves of kernel methods: empirical data versus teacher-student paradigm *J. Stat. Mech.* 124001
[10] Ansuini A, Laio A, Macke J H and Zoccolan D 2019 Intrinsic dimension of data representations in deep neural networks *Advances in Neural Information Processing Systems* vol 32, ed H Wallach, H Larochelle, A Beygelzimer, F d'Alché-Buc, E Fox and R Garnett (Curran Associates, Inc.)
[11] Shwartz-Ziv R and Tishby N 2017 Opening the black box of deep neural networks via (arXiv:1703.00810 [cs])
[12] Recanatesi S, Farrell M, Advani M, Moore T, Lajoie G and Shea-Brown E 2019 Dimensionality compression and expansion in deep neural networks (arXiv:1906.00443 [cs, stat])
[13] Bruna J and Mallat S 2013 Invariant scattering convolution networks *IEEE Trans. Pattern Anal. Mach. Intell.* **35** 1872–86
[14] Mallat S 2016 Understanding deep convolutional networks *Phil. Trans. R. Soc.* A **374** 20150203

[15] Petrini L, Favero A, Geiger M and Wyart M 2021 Relative stability toward diffeomorphisms indicates performance in deep nets *Advances in Neural Information Processing Systems* vol 34 (Curran Associates, Inc.) pp 8727–39
[16] Goodfellow I, Bengio Y and Courville A 2016 *Deep Learning* (The MIT Press)
[17] Hubel D H and Wiesel T N 1962 Receptive fields, binocular interaction and functional architecture in the cat's visual cortex *J. Physiol.* **160** 106–54
[18] Niyogi P, Girosi F and Poggio T 1998 Incorporating prior information in machine learning by creating virtual examples *Proc. IEEE* **86** 2196–209
[19] Anselmi F, Leibo J Z, Rosasco L, Mutch J, Tacchetti A and Poggio T 2016 Unsupervised learning of invariant representations *Theor. Comput. Sci.* **633** 112–21
[20] Poggio T A and Anselmi F 2016 *Visual Cortex and Deep Networks: Learning Invariant Representations* (MIT Press)
[21] Lecun Y, Bottou L, Bengio Y and Haffner P 1998 Gradient-based learning applied to document recognition *Proc. IEEE* **86** 2278–324
[22] Ruderman A, Rabinowitz N C, Morcos A S and Zoran D 2018 Pooling is neither necessary nor sufficient for appropriate deformation stability in CNNs (arXiv:1804.04438 [cs, stat])
[23] Cohen T and Welling M 2014 Learning the irreducible representations of commutative lie groups *Proc. 31st Int. Conf. on Machine Learning* (*Proc. Machine Learning Research* vol 32) ed E P Xing and T Jebara (PMLR) pp 1755–63
[24] Cohen T and Welling M 2016 Group equivariant convolutional networks *Proc. 33rd Int. Conf. on Machine Learning* (*Proc. Machine Learning Research* vol 48) ed M F Balcan and K Q Weinberger (PMLR) pp 2990–9
[25] Ensign D, Neville S, Paul A and Venkatasubramanian S 2017 The complexity of explaining neural networks through (group) invariants *Proc. 28th Int. Conf. on Algorithmic Learning Theory* (*Proc. Machine Learning Research* vol 76) ed S Hanneke and L Reyzin (PMLR) pp 341–59
[26] Kondor R and Trivedi S 2018 On the generalization of equivariance and convolution in neural networks to the action of compact groups *Proc. 35th Int. Conf. on Machine Learning* (*Proc. Machine Learning Research* vol 80) ed J Dy and A Krause (PMLR) pp 2747–55
[27] Finzi M, Welling M and Wilson A G G 2021 A practical method for constructing equivariant multilayer perceptrons for arbitrary matrix groups *Proc. 38th Int. Conf. on Machine Learning* (*Proc. Machine Learning Research* vol 139) ed M Meila and T Zhang (PMLR) pp 3318–28
[28] Blum-Smith B and Villar S 2023 Machine learning and invariant theory (arXiv:2209.14991)
[29] Gregory W, Hogg D W, Blum-Smith B, Arias M T, Wong K W K and Villar S 2023 Geometricimagenet: extending convolutional neural networks to vector and tensor images (arXiv:2305.12585)
[30] Batzner S, Musaelian A, Sun L, Geiger M, Mailoa J P, Kornbluth M, Molinari N, Smidt T E and Kozinsky B 2022 E(3)-equivariant graph neural networks for data-efficient and accurate interatomic potentials *Nat. Commun.* **13** 2453
[31] Saxe A M, Bansal Y, Dapello J, Advani M, Kolchinsky A, Tracey B D and Cox D D 2019 On the information bottleneck theory of deep learning *J. Stat. Mech.* 124020
[32] Fawzi A and Frossard P 2015 Manitest: are classifiers really invariant? *Proc. British Machine Vision Conf. 2015* (British Machine Vision Association) pp 106.1–13
[33] Kanbak C, Moosavi-Dezfooli S M and Frossard P 2018 Geometric robustness of deep networks: analysis and improvement *2018 IEEE/CVF Conf. on Computer Vision and Pattern Recognition* (IEEE) pp 4441–9
[34] Alcorn M A, Li Q, Gong Z, Wang C, Mai L, Ku W S and Nguyen A 2019 Strike (with) a pose: neural networks are easily fooled by strange poses of familiar objects *2019 IEEE/CVF Conf. on Computer Vision and Pattern Recognition* (*CVPR*) (IEEE) pp 4840–9
[35] Alaifari R, Alberti G S and Gauksson T 2018 ADef: an iterative algorithm to construct adversarial deformations (available at: https://openreview.net/forum?id=Hk4dFjR5K7)
[36] Athalye A, Engstrom L, Ilyas A and Kwok K 2018 Synthesizing robust adversarial examples *Int. Conf. on Machine Learning* (PMLR) pp 284–93
[37] Xiao C, Zhu J Y, Li B, He W, Liu M and Song D 2018 Spatially transformed adversarial examples (available at: https://openreview.net/forum?id=HyydRMZC-)
[38] Engstrom L, Tran B, Tsipras D, Schmidt L and Madry A 2019 Exploring the landscape of spatial robustness *Int. Conf. on Machine Learning* (PMLR) pp 1802–11
[39] Paszke A *et al* 2019 PyTorch: an imperative style, high-performance deep learning library *Advances in Neural Information Processing Systems* vol 32
[40] Simonyan K and Zisserman A 2015 Very deep convolutional networks for large-scale image recognition *ICLR* (available at: https://arxiv.org/abs/1409.1556)
[41] Smith S W 1997 *The Scientist and Engineer's Guide to Digital Signal Processing* (California Technical Publishing)
[42] Schoenholz S S, Gilmer J, Ganguli S and Sohl-Dickstein J 2017 Deep information propagation *Int. Conf. on Learning Representations*
[43] Xiao L, Bahri Y, Sohl-Dickstein J, Schoenholz S S and Pennington J 2018 Dynamical isometry and a mean field theory of CNNs: how to train 10,000-layer vanilla convolutional neural networks vol 12 (available at: https://proceedings.mlr.press/v80/xiao18a/xiao18a.pdf)
[44] Risken H 1996 The *Fokker-Planck Equation* (*SSSYN* 18) (Springer)
[45] Saloff-Coste L and Bremaud P 2000 Markov chains: Gibbs fields, Monte Carlo simulation and queues *J. Am. Stat. Assoc.* **95** 1378
[46] LeCun Y, Boser B, Denker J S, Henderson D, Howard R E, Hubbard W and Jackel L D 1989 Backpropagation applied to handwritten zip code recognition *Neural Comput.* **1** 541–51
[47] Krizhevsky A, Sutskever I and Hinton G E 2012 Imagenet classification with deep convolutional neural networks *Advances in Neural Information Processing Systems* vol 25, ed F Pereira, C J C Burges, L Bottou and K Q Weinberger (Curran Associates, Inc.) pp 1097–105
[48] He K, Zhang X, Ren S and Sun J 2016 Deep residual learning for image recognition *2016 IEEE Conf. on Computer Vision and Pattern Recognition* (*CVPR*) pp 770–8
[49] Tan M and Le Q 2019 EfficientNet: rethinking model scaling for convolutional neural networks *Int. Conf. on Machine Learning* (PMLR) pp 6105–14