Input Similarity from the Neural Network Perspective

Guillaume Charpiat[1]   Nicolas Girard[2]   Loris Felardos[1]   Yuliya Tarabalka[2,3]

[1] TAU team, Inria Saclay, LRI, Université Paris-Sud
[2] Université Côte d'Azur, Inria Sophia-Antipolis
[3] LuxCarta Technology
firstname.lastname@inria.fr

Abstract

We first exhibit a multimodal image registration task, for which a neural network trained on a dataset with noisy labels reaches almost perfect accuracy, far beyond noise variance. This surprising auto-denoising phenomenon can be explained as a noise averaging effect over the labels of similar input examples. This effect theoretically grows with the number of similar examples; the question is then to define and estimate the *similarity* of examples.

We express a proper definition of similarity, from the neural network perspective, *i.e.* we quantify how undissociable two inputs A and B are, taking a machine learning viewpoint: how much a parameter variation designed to change the output for A would impact the output for B as well?

We study the mathematical properties of this similarity measure, and show how to use it on a trained network to estimate sample density, in low complexity, enabling new types of statistical analysis for neural networks. We analyze data by retrieving samples perceived as similar by the network, and are able to quantify the denoising effect without requiring true labels. We also propose, during training, to enforce that examples known to be similar should also be seen as similar by the network, and notice speed-up training effects for certain datasets.

1   Motivation: Dataset self-denoising

In remote sensing imagery, data is abundant but noisy [17]. For instance RGB satellite images and binary cadaster maps (delineating buildings) are numerous but badly aligned for various reasons (annotation mistakes, atmosphere disturbance, elevation variations...). In a recent preliminary work [7], we tackled the task of automatically registering these two types of images together with neural networks, considering as ground truth a dataset of hand-picked relatively-well-aligned areas [16], and hoping the network would be able to learn from such a dataset of imperfect alignments. Learning with noisy labels is indeed an active topic of research [24, 18, 15].

For this, we designed an iterative approach: train, then test on the training set and re-align it accordingly; repeat (for 3 iterations). The results were surprisingly good, yielding far better alignments than the ground truth it learned from, both qualitatively (Figure 1) and quantitatively (Figure 2, obtained on manually-aligned data): the median registration error dropped from 18 pixels to 3.5 pixels, which is the best score one could hope for, given intrinsic ambiguities in such registration task. To check that this performance was not due to a subset of the training data that would be perfectly aligned, we added noise to the ground truth and re-trained from it: the new results were about as good again (dashed lines). Thus the network did learn almost perfectly just from noisy labels.

An explanation for this self-denoising phenomenon is proposed in [14] as follows. Let us consider a regression task, with a L^2 loss, and where true labels y were altered with i.i.d. noise ε of variance v.

Figure 1: Qualitative alignment results [7] on a crop of bloomington22 from the Inria dataset [16]. Red: initial dataset annotations; blue: aligned annotations round 1; green: aligned annotations round 2.
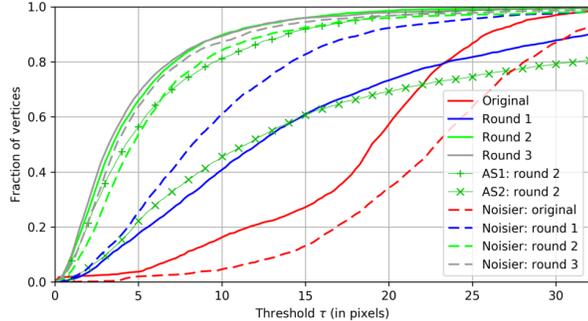

Figure 2: Accuracy cumulative distributions [7] measured with the manually-aligned annotations of bloomington22 [16]. Read as: fraction of image pixels whose registration error is less than threshold τ.

Suppose a same input $\mathbf{x}$ appears n times in the training set, thus with n different labels $y_i = y + \varepsilon_i$. The network can only output the same prediction for all these n cases (since the input is the same), and the best option, considering the L^2 loss, is to predict the average $\frac{1}{n}\sum_i y_i$, whose distance to the true label y is $O(\frac{v}{\sqrt{n}})$. Thus a denoising effect by a factor $\sqrt{n}$ can be observed. However, the exact same point $\mathbf{x}$ is not likely to appear several times in a dataset (with different labels). Rather, relatively *similar* points may appear, and the amplitude of the self-denoising effect will be a function of their number. Here, the similarity should reflect the neural network perception (similar inputs yield the same output) and not an *a priori* norm chosen on the input space.

The purpose of this article is to express the notion of similarity from the network's point of view. We first define it, and study it mathematically, in Section 2, in the one-dimensional output case for the sake of simplicity. Higher-dimensional outputs are dealt with in Section 3. We then compute, in Section 4, the number of neighbors (*i.e.*, of similar samples), and propose for this a very fast estimator. This brings new tools to analyze already-trained networks. As they are differentiable and fast to compute, they can be used during training as well, *e.g.*, to enforce that given examples should be perceived as similar by the network (*c.f.* Section 5). Finally, in Section 6, we apply the proposed tools to analyze a network trained with noisy labels for a remote sensing image alignment task, and formalize the self-denoising phenomenon, quantifying its effect, extending [14] to real datasets.

2   Similarity

2.1   Notions of similarities

The notion of similarity between data points is an important topic in the machine learning literature, obviously in domains such as image retrieval, where images similar to a query have to be found; but not only. For instance when training auto-encoders, the quality of the reconstruction is usually quantified as the L^2 norm between the input and output images. Such a similarity measure is however questionable, as color comparison, performed pixel per pixel, is a poor estimate of human perception: the L^2 norm can vary a lot with transformations barely noticeable to the human eye such as small translations or rotations (for instance on textures), and does not carry semantic information, *i.e.* whether the same kind of objects are present in the image.

Therefore, so-called *perceptual losses* [12] were introduced to quantify image similarity: each image is fed to a standard pre-trained network such as VGG, and the activations in a particular intermediate layer are used as descriptors of the image [5, 6]. The distance between two images is then set as the L^2 norm between these activations. Such a distance carries implicitly semantic information, as the VGG network was trained for image classification. However, the choice of the layer to consider is arbitrary. In the ideal case, one would wish to combine the information from all layers, as some are more abstract and some more detail-specific. But then the particular weights chosen to combine the different layers would also be arbitrary. Would it be possible to get a canonical similarity measure, well posed theoretically?

2

More importantly, the previous litterature does not consider the notion of input similarity from the point of view of the neural network that is being used, but from the point of view of another one (typically, VGG) which aims at imitating human perception. A notable exception [13] transposes to machine learning the concept of influence functions in statistics [9]. The differences with our definition of similarity might seem slight at first glance but they have important consequences: first, making use of the loss (and of its gradient and its Hessian) in the similarity measure has the issue that the expressed quantities are not intrinsic to the neural network but also depend on the optimization criterion used during training, which is problematic in the case of noisy labels as, at training convergence, the gradient of the loss with respect to the output points in random directions (remaining label noise that the network is not able to overfit). Second, the inverse of the Hessian appears in influence functions, while our definition makes use of gradients only. Another interesting related work [25] expresses neural networks as a kernel between test point and training points. Once again however the kernel definition relies on the training criterion.

As a supplementary motivation for this study, neural networks are black boxes difficult to interpret, and showing which samples a network considers as similar would help to explain its decisions. Also, the number of such similar examples would be a key element for confidence estimation at test time.

In this section we define a proper, intrinsic notion of similarity as seen by the network, relying on how easily it can distinguish different inputs.

2.2 Similarity from the point of view of the parameterized family of functions

Let f_θ be a parameterized function, typically a neural network already trained for some task, and $\mathbf{x}$, $\mathbf{x}'$ possible inputs, for instance from the training or test set. For the sake of simplicity, let us suppose in a first step that f_θ is real valued. To express the similarity between $\mathbf{x}$ and $\mathbf{x}'$, as seen by the network, one could compare the output values $f_\theta(\mathbf{x})$ and $f_\theta(\mathbf{x}')$. This is however not very informative, and a same output might be obtained for different reasons.

Instead, we define similarity as the influence of $\mathbf{x}$ over $\mathbf{x}'$, by quantifying how much an additional training step for $\mathbf{x}$ would change the output for $\mathbf{x}'$ as well. If $\mathbf{x}$ and $\mathbf{x}'$ are very different from the point of view of the neural network, changing $f_\theta(\mathbf{x})$ will have little consequence on $f_\theta(\mathbf{x}')$. Vice versa, if they are very similar, changing $f_\theta(\mathbf{x})$ will greatly affect $f_\theta(\mathbf{x}')$ as well.
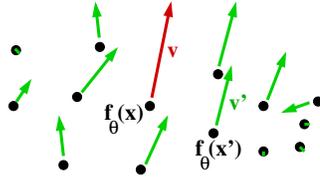

Figure 3: Moves in the space of outputs. We quantify the influence of a data point $\mathbf{x}$ over another one $\mathbf{x}'$ by how much the tuning of parameters θ to obtain a desired output change $\mathbf{v}$ for $f_\theta(\mathbf{x})$ will affect $f_\theta(\mathbf{x}')$ as well.

Formally, if one wants to change the value of $f_\theta(\mathbf{x})$ by a small quantity ε, one needs to update θ by $\delta\theta = \varepsilon \frac{\nabla_\theta f_\theta(\mathbf{x})}{\|\nabla_\theta f_\theta(\mathbf{x})\|^2}$. Indeed, after the parameter update, the new value at $\mathbf{x}$ will be:

$$f_{\theta+\delta\theta}(\mathbf{x}) = f_\theta(\mathbf{x}) + \nabla_\theta f_\theta(\mathbf{x}) \cdot \delta\theta + O(\|\delta\theta\|^2) = f_\theta(\mathbf{x}) + \varepsilon + O(\varepsilon^2).$$

This parameter change induces a value change at any other point $\mathbf{x}'$:

$$f_{\theta+\delta\theta}(\mathbf{x}') = f_\theta(\mathbf{x}') + \nabla_\theta f_\theta(\mathbf{x}') \cdot \delta\theta + O(\|\delta\theta\|^2) = f_\theta(\mathbf{x}') + \varepsilon \frac{\nabla_\theta f_\theta(\mathbf{x}') \cdot \nabla_\theta f_\theta(\mathbf{x})}{\|\nabla_\theta f_\theta(\mathbf{x})\|^2} + O(\varepsilon^2).$$

Therefore the kernel $k_\theta^N(\mathbf{x}, \mathbf{x}') = \dfrac{\nabla_\theta f_\theta(\mathbf{x}) \cdot \nabla_\theta f_\theta(\mathbf{x}')}{\|\nabla_\theta f_\theta(\mathbf{x})\|^2}$ represents the influence of $\mathbf{x}$ over $\mathbf{x}'$: if one wishes to change the output value $f_\theta(\mathbf{x})$ by ε, then $f_\theta(\mathbf{x}')$ will change by $\varepsilon\, k_\theta^N(\mathbf{x}, \mathbf{x}')$. In particular, if $k_\theta^N(\mathbf{x}, \mathbf{x}')$ is high, then $\mathbf{x}$ and $\mathbf{x}'$ are not distinguishable from the point of view of the network, as any attempt to move $f_\theta(\mathbf{x})$ will move $f_\theta(\mathbf{x}')$ as well (see Fig. 3). We thus see $k_\theta^N(\mathbf{x}, \mathbf{x}')$ as a measure of similarity. Note however that $k_\theta^N(\mathbf{x}, \mathbf{x}')$ is not symmetric.

Symmetric similarity: correlation   Two symmetric kernels natural arise: the inner product:

$$k_\theta^I(\mathbf{x}, \mathbf{x}') \;=\; \nabla_\theta f_\theta(\mathbf{x}) \cdot \nabla_\theta f_\theta(\mathbf{x}') \tag{1}$$

and its normalized version, the correlation:

$$k_\theta^C(\mathbf{x}, \mathbf{x}') = \frac{\nabla_\theta f_\theta(\mathbf{x})}{\|\nabla_\theta f_\theta(\mathbf{x})\|} \cdot \frac{\nabla_\theta f_\theta(\mathbf{x}')}{\|\nabla_\theta f_\theta(\mathbf{x}')\|} \tag{2}$$

which has the advantage of being bounded (in $[-1, 1]$), thus expressing similarity in a usual meaning.

2.3   Properties for vanilla neural networks

Intuitively, inputs that are similar from the network perspective should produce similar outputs; we can check that k_θ^C is a good similarity measure in this respect (all proofs are deferred to the Appendix):

Theorem 1. *For any real-valued neural network f_θ whose last layer is a linear layer (without any parameter sharing) or a standard activation function thereof (sigmoid, tanh, ReLU...), and for any inputs $\mathbf{x}$ and $\mathbf{x}'$,*

$$\nabla_\theta f_\theta(\mathbf{x}) = \nabla_\theta f_\theta(\mathbf{x}') \quad \Longrightarrow \quad f_\theta(\mathbf{x}) = f_\theta(\mathbf{x}')\,.$$

Corollary 1. *Under the same assumptions, for any inputs $\mathbf{x}$ and $\mathbf{x}'$,*

$$k_\theta^C(\mathbf{x}, \mathbf{x}') = 1 \quad \Longrightarrow \quad \nabla_\theta f_\theta(\mathbf{x}) = \nabla_\theta f_\theta(\mathbf{x}')\,,$$
$$\text{hence} \quad k_\theta^C(\mathbf{x}, \mathbf{x}') = 1 \quad \Longrightarrow \quad f_\theta(\mathbf{x}) = f_\theta(\mathbf{x}')\,.$$

Furthermore,

Theorem 2. *For any real-valued neural network f_θ without parameter sharing, if $\nabla_\theta f_\theta(\mathbf{x}) = \nabla_\theta f_\theta(\mathbf{x}')$ for two inputs $\mathbf{x}, \mathbf{x}'$, then all useful activities computed when processing $\mathbf{x}$ are equal to the ones obtained when processing $\mathbf{x}'$.*

We name *useful* activities all activities $a_i(\mathbf{x})$ whose variation would have an impact on the output, *i.e.* all the ones satisfying $\frac{df_\theta(\mathbf{x})}{da_i} \neq 0$. This condition is typically not satisfied when the activity is negative and followed by a ReLU, or when it is multiplied by a 0 weight, or when all its contributions to the output cancel one another (*e.g.*, a sum of two neurons with opposite weights: $f_\theta(\mathbf{x}) = \sigma(a_i(\mathbf{x})) - \sigma(a_i(\mathbf{x}))$).

Link with the *perceptual loss*   For a vanilla network without parameter sharing, the gradient $\nabla_\theta f_\theta(\mathbf{x})$ is a list of coefficients $\nabla_{w_i^j} f_\theta(\mathbf{x}) = \frac{df_\theta(\mathbf{x})}{db_j} a_i(\mathbf{x})$, where w_i^j is the parameter-factor that multiplies the input activation $a_i(\mathbf{x})$ in neuron j, and of coefficients $\nabla_{b_j} f_\theta(\mathbf{x}) = \frac{df_\theta(\mathbf{x})}{db_j}$ for neuron biases, which we will consider as standard parameters $b_j = w_0^j$ that act on a constant activation $a_0(\mathbf{x}) = 1$, yielding $\nabla_{w_0^j} f_\theta(\mathbf{x}) = \frac{df_\theta(\mathbf{x})}{db_j} a_0(\mathbf{x})$. Thus the gradient $\nabla_\theta f_\theta(\mathbf{x})$ can be seen as a list of all activation values $a_i(\mathbf{x})$ multiplied by the potential impact on the output $f_\theta(\mathbf{x})$ of the neurons j using them, *i.e.* $\frac{df_\theta(\mathbf{x})}{db_j}$. Each activation appears in this list as many times as it is fed to different neurons. The similarity between two inputs then rewrites:

$$k_\theta^I(\mathbf{x}, \mathbf{x}') = \sum_{\text{activities } i} \lambda_i(\mathbf{x}, \mathbf{x}')\, a_i(\mathbf{x})\, a_i(\mathbf{x}') \quad \text{where} \quad \lambda_i(\mathbf{x}, \mathbf{x}') = \sum_{\text{neuron } j \text{ using } a_i} \frac{df_\theta(\mathbf{x})}{db_j} \frac{df_\theta(\mathbf{x}')}{db_j}$$

are data-dependent importance weights. Such weighting schemes on activation units naturally arise when expressing intrinsic quantities; the use of natural gradients would bring invariance to re-parameterization [19, 20]. On the other hand, the inner product related to the perceptual loss would be

$$\sum_{\text{activities } i \neq 0} \lambda_{\text{layer}(i)}\, a_i(\mathbf{x})\, a_i(\mathbf{x}')$$

for some arbitrary fixed layer-dependent weights $\lambda_{\text{layer}(i)}$.

2.4 Properties for parameter-sharing networks

When sharing weights, as in convolutional networks, the gradient $\nabla_\theta f_\theta(\mathbf{x})$ is made of the same coefficients (impact-weighted activations) but summed over shared parameters. Denoting by $\mathcal{S}(i)$ the set of (neuron, input activity) pairs where the parameter w_i is involved,

$$k_\theta^I(\mathbf{x}, \mathbf{x}') = \sum_{\text{params } i} \left(\sum_{(j,k) \in \mathcal{S}_i} a_k(\mathbf{x}) \frac{df_\theta(\mathbf{x})}{db_j} \right) \left(\sum_{(j,k) \in \mathcal{S}_i} a_k(\mathbf{x}') \frac{df_\theta(\mathbf{x}')}{db_j} \right)$$

Thus, in convolutional networks, k_θ^I similarity does not imply similarity of first layer activations anymore, but only of their (impact-weighted) spatial average. More generally, any invariance introduced by a weight sharing scheme in an architecture will be reflected in the similarity measure k_θ^I, which is expected as k_θ^I was defined as the input similarity *from the neural network perspective*.

Note that this type of objects was recently studied from an optimization viewpoint under the name of Neural Tangent Kernel [11, 2] in the infinite layer width limit.

3   Higher output dimension

Let us now study the more complex case where $f_\theta(\mathbf{x})$ is a vector $\left(f_\theta^i(\mathbf{x}) \right)_{i \in [1,d]}$ in $\mathbb{R}^d$ with $d > 1$. Under a mild hypothesis on the network (output expressivity), always satisfied unless specially designed not to:

Theorem 3. *The optimal parameter change $\delta\theta$ to push $f_\theta(\mathbf{x})$ in a direction $\mathbf{v} \in \mathbb{R}^d$ (with a force $\varepsilon \in \mathbb{R}$), i.e. such that $f_{\theta+\delta\theta}(\mathbf{x}) - f_\theta(\mathbf{x}) = \varepsilon\mathbf{v}$, induces at any other point $\mathbf{x}'$ the following output variation:*

$$f_{\theta+\delta\theta}(\mathbf{x}') - f_\theta(\mathbf{x}') = \varepsilon\, K_\theta(\mathbf{x}', \mathbf{x})\, K_\theta(\mathbf{x}, \mathbf{x})^{-1}\, \mathbf{v} + O(\varepsilon^2) \qquad (3)$$

where the $d \times d$ kernel matrix $K_\theta(\mathbf{x}', \mathbf{x})$ is defined by $K_\theta^{ij}(\mathbf{x}', \mathbf{x}) = \nabla_\theta f_\theta^i(\mathbf{x}') \cdot \nabla_\theta f_\theta^j(\mathbf{x})$.

The similarity kernel is now a matrix and not just a single value, as it describes the relation between moves $\mathbf{v} \in \mathbb{R}^d$. Note that these matrices K_θ are only $d \times d$ where d is the output dimension. They are thus generally small and easy to manipulate or inverse.

Normalized similarity matrix   The unitless symmetrized, normalized version of the kernel (3) is:

$$K_\theta^C(\mathbf{x}, \mathbf{x}') = K_\theta(\mathbf{x}, \mathbf{x})^{-1/2}\, K_\theta(\mathbf{x}, \mathbf{x}')\, K_\theta(\mathbf{x}', \mathbf{x}')^{-1/2} . \qquad (4)$$

It has the following properties: its coefficients are bounded, in $[-1, 1]$; its trace is at most d; its (Frobenius) norm is at most $\sqrt{d}$; self-similarity is identity: $\forall \mathbf{x},\ K_\theta^C(\mathbf{x}, \mathbf{x}) = \text{Id}$; the kernel is symmetric, in the sense that $K_\theta^C(\mathbf{x}', \mathbf{x}) = K_\theta^C(\mathbf{x}, \mathbf{x}')^T$.

Similarity in a single value   To summarize the similarity matrix $K_\theta^C(\mathbf{x}, \mathbf{x}')$ into a single real value in $[-1, 1]$, we consider:

$$k_\theta^C(\mathbf{x}, \mathbf{x}') = \frac{1}{d} \text{Tr}\, K_\theta^C(\mathbf{x}, \mathbf{x}') . \qquad (5)$$

It can be shown indeed that if $k_\theta^C(\mathbf{x}, \mathbf{x}')$ is close to 1, then $K_\theta^C(\mathbf{x}, \mathbf{x}')$ is close to Id, and reciprocally. See Appendix C for more details and a discussion about the links between $\frac{1}{d} \text{Tr}\, K_\theta^C(\mathbf{x}, \mathbf{x}')$ and $\left\| K_\theta^C(\mathbf{x}, \mathbf{x}') - \text{Id} \right\|_F$.

Metrics on output: rotation invariance   Similarity in $\mathbb{R}^d$ might be richer than just estimating distances in L^2 norm. For instance, for our 2D image registration task, the network could be known (or desired) to be equivariant to rotations. The similarity between two output variations $\mathbf{v}$ and $\mathbf{v}'$ can be made rotation-invariant by applying the rotation that best aligns $\mathbf{v}$ and $\mathbf{v}'$ beforehand. This can actually be easily computed in closed form and yields:

$$k_\theta^{C,\text{rot}}(\mathbf{x}, \mathbf{x}') = \frac{1}{2} \sqrt{\left\| K_\theta^C(\mathbf{x}, \mathbf{x}') \right\|_F^2 + 2 \det K_\theta^C(\mathbf{x}, \mathbf{x}')} .$$

Note that other metrics are possible in the output space. For instance, the loss metric quantifies the norm of a move $\mathbf{v}$ by its impact on the loss $\left. \frac{dL(y)}{dy} \right|_{f_\theta(\mathbf{x})}(\mathbf{v})$. It has a particular meaning though, is not intrinsic, and is not always relevant, *e.g.* in the noisy label case seen in Section 1.

The case of classification tasks When the output of the network is a probability distribution $p_{\theta,\mathbf{x}}(c)$, over a finite number of given classes c for example, it is natural from an information theoretic point of view to rather consider $f_\theta^c(\mathbf{x}) = -\log p_{\theta,\mathbf{x}}(c)$. This is actually the quantities computed in the pre-softmax layer from which common practice directly computes the cross-entropy loss.

It turns out that the L^2 norm of variations δf in this space naturally corresponds to the Fisher information metric, which quantifies the impact of parameter variations $\delta\theta$ on the output probability $p_{\theta,\mathbf{x}}$, as $\mathrm{KL}(p_{\theta,\mathbf{x}}||p_{\theta+\delta\theta,\mathbf{x}})$. The matrices $K_\theta(\mathbf{x},\mathbf{x}) = \left(\nabla_\theta f_\theta^c(\mathbf{x}) \cdot \nabla_\theta f_\theta^{c'}(\mathbf{x})\right)_{c,c'}$ and $F_{\theta,\mathbf{x}} = \mathbb{E}_c\left[\nabla_\theta f_\theta^c(\mathbf{x}) \nabla_\theta f_\theta^c(\mathbf{x})^T\right]$ are indeed to each other what correlation is to covariance. Thus the quantities defined in Equation (5) already take into account information geometry when applied to the pre-softmax layer, and do not need supplementary metric adjustment.

Faster setup for classification tasks with many classes In a classification task in d classes with large d, the computation of $d \times d$ matrices may be prohibitive. As a workaround, for a given input training sample $\mathbf{x}$, the classification task can be seen as a binary one (the right label c_R *vs.* the other ones), in which case the d outputs of the neural network can be accordingly combined in a single real value. The 1D similarity measure can then be used to compare any training samples of the same class.

When making statistics on similarity values $\mathbb{E}_{\mathbf{x}'}\left[k_\theta^C(\mathbf{x},\mathbf{x}')\right]$, another possible task binarization approach is to sample an adversary class c_A along with $\mathbf{x}'$, and hence consider $\nabla_\theta f_\theta^{c_R}(\mathbf{x}) - \nabla_\theta f_\theta^{c_A}(\mathbf{x})$. Both approaches will lead to similar results in Section 5.

4 Estimating density

In this section, we use similarity to estimate input neighborhoods and perform statistics on them.

4.1 Estimating the number of neighbors

Given a point $\mathbf{x}$, how many samples $\mathbf{x}'$ are similar to $\mathbf{x}$ according to the network? This can be measured by computing $k_\theta^C(\mathbf{x},\mathbf{x}')$ for all $\mathbf{x}'$ and picking the closest ones, *i.e. e.g.* the $\mathbf{x}'$ such that $k_\theta^C(\mathbf{x},\mathbf{x}') \geqslant 0.9$. More generally, for any data point $\mathbf{x}$, the histogram of the similarity $k_\theta^C(\mathbf{x},\mathbf{x}')$ over all $\mathbf{x}'$ in the dataset (or a representative subset thereof) can be drawn, and turned into an estimate of the number of neighbors of $\mathbf{x}$. To do this, several types of estimates are possible:

- hard-thresholding, for a given threshold $\tau \in [0,1]$: $\qquad N_\tau(\mathbf{x}) = \sum_{\mathbf{x}'} \mathbb{1}_{k_\theta^C(\mathbf{x},\mathbf{x}') \geqslant \tau}$
- soft estimate: $\qquad\qquad\qquad\qquad\qquad\qquad\qquad N_S(\mathbf{x}) = \sum_{\mathbf{x}'} k_\theta^C(\mathbf{x},\mathbf{x}')$
- less-soft positive-only estimate ($\alpha > 0$): $\qquad N_\alpha^+(\mathbf{x}) = \sum_{\mathbf{x}'} \mathbb{1}_{k_\theta^C(\mathbf{x},\mathbf{x}')>0}\, k_\theta^C(\mathbf{x},\mathbf{x}')^\alpha$

In practice we observe that k_θ^C is very rarely negative, and thus the soft estimate N_S can be justified as an average of the hard-thresholding estimate N_τ over all possible thresholds τ:

$$\int_{\tau=0}^{1} N_\tau(\mathbf{x}) d\tau = \sum_{\mathbf{x}'} \int_{\tau=0}^{1} \mathbb{1}_{k_\theta^C(\mathbf{x},\mathbf{x}') \geqslant \tau}\, d\tau = \sum_{\mathbf{x}'} k_\theta^C(\mathbf{x},\mathbf{x}')\, \mathbb{1}_{k_\theta^C(\mathbf{x},\mathbf{x}') \geqslant 0} = N_1^+(\mathbf{x}) \simeq N_S(\mathbf{x})$$

4.2 Low complexity of the soft estimate $N_S(\mathbf{x})$

The soft estimate $N_S(\mathbf{x})$ is rewritable as:

$$\sum_{\mathbf{x}'} k_\theta^C(\mathbf{x},\mathbf{x}') = \sum_{\mathbf{x}'} \frac{\nabla_\theta f_\theta(\mathbf{x})}{\|\nabla_\theta f_\theta(\mathbf{x})\|} \cdot \frac{\nabla_\theta f_\theta(\mathbf{x}')}{\|\nabla_\theta f_\theta(\mathbf{x}')\|} = \frac{\nabla_\theta f_\theta(\mathbf{x})}{\|\nabla_\theta f_\theta(\mathbf{x})\|} \cdot \mathbf{g} \quad \text{with} \quad \mathbf{g} = \sum_{\mathbf{x}'} \frac{\nabla_\theta f_\theta(\mathbf{x}')}{\|\nabla_\theta f_\theta(\mathbf{x}')\|}$$

and consequently $N_S(\mathbf{x})$ can be computed jointly for all $\mathbf{x}$ in linear time $O(|\mathcal{D}|p)$ in the dataset size $|\mathcal{D}|$ and in the number of parameters p, in just two passes over the dataset, when the output dimension is 1. For higher output dimensions d, a similar trick can be used and the complexity becomes $O(|\mathcal{D}|d^2p)$. For classification tasks with a large number d of classes, the complexity can be reduced to $O(|\mathcal{D}|p)$ through an approximation consisting in binarizing the task (*c.f.* end of Section 3).
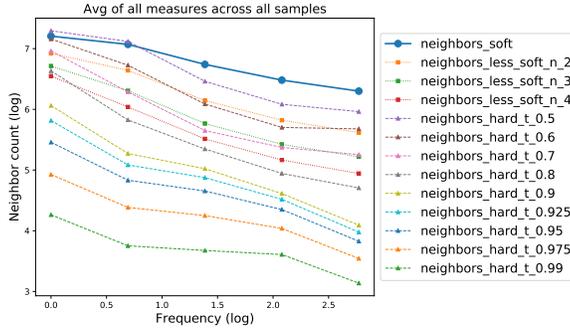
Figure 4: Density estimation using the various approaches (log scale). All approaches behave similarly and show good results, except the ones with extreme thresholds.
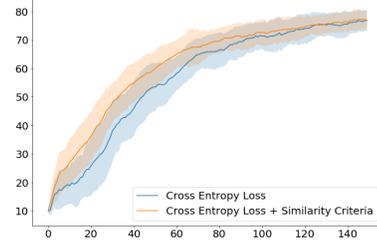

Figure 5: Validation accuracy of a neural network trained on MNIST with and without the similarity criterion (note that the x-axis is the number of minibatches presented to the network, not of epochs).

4.3 Test of the various estimators

In order to rapidly test the behavior of all possible estimators, we applied them to a toy problem where the network's goal is to predict a sinusoid. To change the difficulty of the problem, we vary its frequency, while keeping the number of samples constant. Appendix D gives more details and results for the toy problem. Fig.4 shows for each estimator (with different parameters when relevant), the result of their neighbor count estimation. When the frequency f of the sinusoid to predict increases, the number of neighbors decreases in $\frac{1}{f}$ for every estimator. This aligns with our intuition that as the problem gets harder, the network needs to distinguish input samples more to achieve a good performance, thus the amount of neighbors is lower. In particular we observe that the proposed $N_S(\mathbf{x})$ estimator behaves well, thus we will use that one in bigger studies requiring an efficient estimator.

4.4 Further potential uses for fitness estimation

When the number of neighbors of a training point $\mathbf{x}$ is very low, the network is able to set any label to $\mathbf{x}$, as this won't interfere with other points, by definition of our similarity criterion $k_\theta(\mathbf{x}, \mathbf{x}')$. This is thus a typical overfit case, where the network can learn by heart a label associated to a particular, isolated point.

On the opposite, when the set of neighbors of $\mathbf{x}$ is a large fraction of the dataset, comprising varied elements, by definition of $k_\theta(\mathbf{x}, \mathbf{x}')$ the network is not able to distinguish them, and consequently it can only provide a common output for all of them. Therefore it might not be able to express variety enough, which would be a typical underfit case.

The quality of fit can thus be observed by monitoring the number of neighbors together with the variance of the desired labels in the neighborhoods (to distinguish underfit from just high density).

Prediction uncertainty A measure of the uncertainty of a prediction $f_\theta(\mathbf{x})$ could be to check how easy it would have been to obtain another value during training, without disturbing the training of other points. A given change $\mathbf{v}$ of $f_\theta(\mathbf{x})$ induces changes $\frac{k_\theta^I(\mathbf{x}, \mathbf{x}')}{\|\nabla_\theta f_\theta(\mathbf{x})\|^2} \mathbf{v}$ over other points $\mathbf{x}'$ of the dataset, creating a total L^1 disturbance $\sum_{\mathbf{x}'} \|\frac{k_\theta^I(\mathbf{x}, \mathbf{x}')}{\|\nabla_\theta f_\theta(\mathbf{x})\|^2} \mathbf{v}\|$. The uncertainty factor would then be the norm of $\mathbf{v}$ affordable within a disturbance level, and quickly approximable as $\frac{\|\nabla_\theta f_\theta(\mathbf{x})\|^2}{\sum_{\mathbf{x}'} k_\theta^I(\mathbf{x}, \mathbf{x}')}$.

5 Enforcing similarity

The similarity criterion we defined could be used not only to estimate how similar two samples are perceived, after training, but also to incite the network, during training, to evolve in order to consider these samples as similar.

7

Asking two samples to be treated as similar   If two inputs $\mathbf{x}$ and $\mathbf{x}'$ are known to be similar (from a human point of view), one can enforce their similarity from the network perspective, by adding to the loss the term:
$$-k_\theta^C(\mathbf{x}, \mathbf{x}') \ .$$

Asking a distribution of samples to be treated as similar   By extension, to enforce the similarity of a subset $\mathcal{S}$ of training samples, of size $n = |\mathcal{S}|$, one might consider the average pairwise similarity k_θ^C over all pairs, or the standard deviation of the gradients. Both turn out to be equivalent to maximizing the norm of the gradient mean $\mu = \frac{1}{n} \sum_{i \in \mathcal{S}} \frac{\nabla_\theta f_\theta(\mathbf{x}_i)}{\|\nabla_\theta f_\theta(\mathbf{x}_i)\|}$:

$$\frac{1}{n(n-1)} \sum_{i,j \in \mathcal{S}, i \neq j} k_\theta^C(x_i, x_j) \ = \ \frac{n}{n-1}\|\mu\|^2 - \frac{1}{n-1} \quad \text{and} \quad \underset{i \in \S}{\text{var}} \frac{\nabla_\theta f_\theta(\mathbf{x}_i)}{\|\nabla_\theta f_\theta(\mathbf{x}_i)\|} = 1 - \|\mu\|^2 \ .$$

In practice, common deep learning platforms are much faster when using mini-batches, but then return only the gradient sum $\sum_{i \in \mathcal{B}} \nabla_\theta f_\theta(\mathbf{x}_i)$ over a mini-batch $\mathcal{B}$, not individual gradients, preventing the normalization of each of them to compute k_θ^C or μ. So instead we compare means of un-normalized gradients, over two mini-batches $\mathcal{B}_1$ and $\mathcal{B}_2$ comprising each n_B samples from $\mathcal{S}$, which yields the criterion:

$$n_B \frac{\|\mu_1 - \mu_2\|^2}{\|\mu_1\|\|\mu_2\|} \quad \text{where} \quad \mu_k = \frac{1}{n} \sum_{i \in \mathcal{B}_k} \nabla_\theta f_\theta(\mathbf{x}_i) \ .$$

The factor n_B counterbalances the $\frac{1}{\sqrt{n_B}}$ variance reduction effect due to averaging over n_B samples.

Group invariance   The distributions of samples asked to be seen as similar could be group orbits [3]. A differential formulation of group invariance enforcement is also proposed in Appendix E.2.

Complexity   The *double-backpropagation* routine, available on common deep learning platforms, allows the optimization of such criteria [4, 10, 22, 8], roughly doubling the computational time of a gradient step.

Dynamics of learning   Our approach enforces similarity not just at the output level, but within the whole internal computational process. Therefore, during training, information is provided directly to each parameter instead of being back-propagated through possibly many layers. Thus the dynamics of learning are expected to be different, especially for deep networks.

To test this hypothesis, we train a small network on MNIST with and without the similarity criteria acting as an auxiliary loss (see Fig. 5). As a result, we observe an acceleration of the convergence very early in the learning process. It is worth noting that this effect can be observed across a wide range of different neural architectures. We performed additional experiments on toy datasets as well as on CIFAR10 with no or only negligible improvements. All together this suggests that using the similarity criteria during training may be beneficial to specific datasets as opposed to specific architectures, and indeed, as the class intra-variability in CIFAR10 is known to be high, considering all examples of a class of CIFAR10 as similar is less relevant.

6   Dataset self-denoising

We now go back to the task described in Section 1 and show how input similarity can be used to analyse experimental results and bring theoretical guarantees about robustness to label noise.

6.1   Similarity experimentally observed between patches

We studied the multi-round training scheme of [7] by applying our similarity measure to a sampling of input patches of the training dataset for one network per round. The principle of the multiple round training scheme is to reduce the noise of the annotations, obtaining aligned annotations in the end (more details in Appendix F). For a certain input patch, we computed its similarity with all the other patches for the 3 networks. With those similarities we can compute the nearest neighbors of that patch, see Fig. 6. The input patch is of a suburb area with sparse houses and individual trees. The closest neighbors look similar as they usually feature the same types of buildings, building

Figure 6: Example of nearest neighbors for a patch. Each line corresponds to a round. Each patch has its similarity written under it.
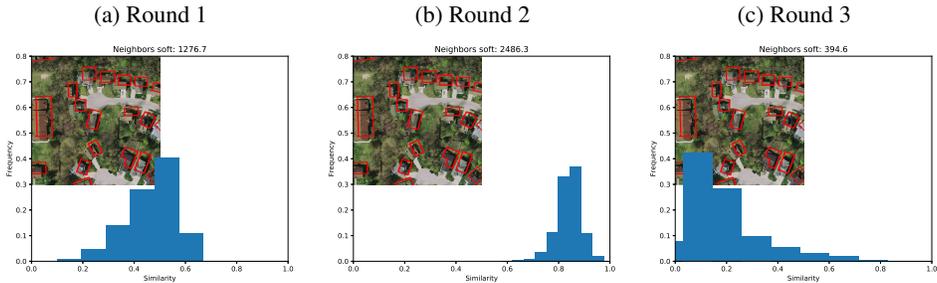

Figure 7: Histograms of similarities for one patch across rounds.

arrangement and vegetation. However sometimes the network sees a patch as similar when it is not clear from our point of view (for example patches with large buildings).

For more in-depth results, we computed the histogram of similarities for the same patch, see Fig. 7. We observe that round 2 shows different neighborhood statistics, in that the patch is closer to all other patches than in other rounds. We observe the same behavior in 19 other input patches (see Appendix F). An hypothesis for this phenomenon is that the average gradient was not 0 at the end of that training round (due to optimization convergence issues, e.g.), which would shift all similarity histograms by a same value.

Qualitatively, for patches randomly sampled, their similarity histograms tend to be approximately symmetric in round 2, but with a longer left tail in round 1 and a longer right tail in round 3. Neighborhoods thus seem to change across the rounds, with fewer and fewer close points (if removing the global histogram shift in round 2). A possible interpretation is that this would reflect an increasing ability of the network to distinguish between different patches, with finer features in later training rounds.

6.2 Comparison to the *perceptual loss*

We compare our approach to the *perceptual loss* on a nearest neighbor retrieval task. We notice that the *perceptual loss* sometimes performs reasonably well, but often not. For instance, we show in Fig. 8 the closest neighbors to a structured residential area image, for the *perceptual loss* (first row: not making sense) and for our similarity measure (second row: similar areas).

6.3 From similarity statistics to self-denoising effect estimation

We now show how such similarity experimental computations can be used to solve the initial problem of Section 1, by explicitly turning similarity statistics into a quantification of the self-denoising effect.

Let us denote by y_i the true (unknown) label for input $\mathbf{x}_i$, by $\widetilde{y}_i$ the noisy label given in the dataset, and by $\widehat{y}_i = f_\theta(\mathbf{x}_i)$ the label predicted by the network. We will denote the (unknown) noise by $\varepsilon_i = \widetilde{y}_i - y_i$ and assume it is centered and i.i.d., with finite variance σ_ε. The training criterion

9

Figure 8: Closest neighbors to the leftmost patch, using the *perceptual loss* (first row) and our similarity definition (second row).

is $E(\theta) = \sum_j ||\widehat{y}_j - \widetilde{y}_j||^2$. At convergence, the training leads to a local optimum of the energy landscape: $\nabla_\theta E = 0$, that is, $\sum_j (\widehat{y}_j - \widetilde{y}_j) \nabla_\theta \widehat{y}_j = 0$. Let's choose any sample i and multiply by $\nabla_\theta \widehat{y}_i$: using $k_\theta^I(\mathbf{x}_i, \mathbf{x}_j) = \nabla_\theta \widehat{y}_i . \nabla_\theta \widehat{y}_j$, we get:

$$\sum_j (\widehat{y}_j - \widetilde{y}_j) \, k_\theta^I(\mathbf{x}_j, \mathbf{x}_i) = 0.$$

Let us denote by $k_\theta^{IN}(\mathbf{x}_j, \mathbf{x}_i) = k_\theta^I(\mathbf{x}_j, \mathbf{x}_i) \big(\sum_j k_\theta^I(\mathbf{x}_j, \mathbf{x}_i) \big)^{-1}$ the column-normalized kernel, and by $\mathbb{E}_k[a] = \sum_j a_j \, k_\theta^{IN}(\mathbf{x}_j, \mathbf{x}_i)$ the mean value of a in the neighborhood of i, that is, the weighted average of the a_j with weights $k_\theta^I(\mathbf{x}_j, \mathbf{x}_i)$ normalized to sum up to 1. This is actually a kernel regression, in the spirit of Parzen-Rosenblatt window estimators. Then the previous property can be rewritten as $\mathbb{E}_k[\widehat{y}] = \mathbb{E}_k[\widetilde{y}]$. As $\mathbb{E}_k[\widetilde{y}] = \mathbb{E}_k[y] + \mathbb{E}_k[\varepsilon]$, this yields:

$$\widehat{y}_i - \mathbb{E}_k[y] = \mathbb{E}_k[\varepsilon] + (\widehat{y}_i - \mathbb{E}_k[\widehat{y}])$$

i.e. the difference between the predicted $\widehat{y}_i$ and the average of the true labels in the neighborhood of i is equal to the average of the noise in the neighborhood of i, up to the deviation of the prediction $\widehat{y}_i$ from the average prediction in its neighborhood.

We want to bound the error $\|\widehat{y}_i - \mathbb{E}_k[y]\|$ without knowing neither the true labels y nor the noise ε. One can show that $\mathbb{E}_k[\varepsilon] \propto \mathrm{var}_\varepsilon(\mathbb{E}_k[\varepsilon])^{1/2} = \sigma_\varepsilon \, \|k_\theta^{IN}(\cdot, \mathbf{x}_i)\|_{L2}$. The denoising factor is thus the similarity kernel norm $\|k_\theta^{IN}(\cdot, \mathbf{x}_i)\|_{L2}$, which is between $1/\sqrt{N}$ and 1, depending on the neighborhood quality. It is $1/\sqrt{N}$ when all N data points are identical, i.e. all satisfying $k_\theta^C(\mathbf{x}_i, \mathbf{x}_j) = 1$. On the other extreme, this factor is 1 when all points are independent: $k_\theta^I(\mathbf{x}_i, \mathbf{x}_j) = 0 \quad \forall i \neq j$. This way we extend *noise2noise* [14] to real datasets with non-identical inputs.

In our remote sensing experiment, we estimate this way a denoising factor of 0.02, consistent across all training rounds and inputs ($\pm 10\%$), implying that each training round contributed equally to denoising the labels. This is confirmed by Fig. 2, which shows the error steadily decreasing, on a control test where true labels are known. The shift $(\widehat{y}_i - \mathbb{E}_k[\widehat{y}])$ on the other hand can be directly estimated given the network prediction. In our case, it is 4.4px on average, which is close to the observed median error for the last round in Fig. 2. It is largely input-dependent, with variance 3.2px, which is reflected by the spread distribution of errors in Fig. 2. This input-dependent shift thus provides a hint about prediction reliability.

It is also possible to bound $(\widehat{y}_i - \mathbb{E}_k[\widehat{y}]) = \mathbb{E}_k[\widehat{y}_i - \widehat{y}]$ using only similarity information (without predictions $\widehat{y}$). Theorem 1 implies that the application: $\frac{\nabla_\theta f_\theta(\mathbf{x})}{\|\nabla_\theta f_\theta(\mathbf{x})\|} \mapsto f_\theta(\mathbf{x})$ is well-defined, and it can actually be shown to be Lipschitz with a network-dependent constant (under mild hypotheses). Thus

$$\|f_\theta(\mathbf{x}) - f_\theta(\mathbf{x}')\| \leqslant C \left\| \frac{\nabla_\theta f_\theta(\mathbf{x})}{\|\nabla_\theta f_\theta(\mathbf{x})\|} - \frac{\nabla_\theta f_\theta(\mathbf{x}')}{\|\nabla_\theta f_\theta(\mathbf{x}')\|} \right\| = \sqrt{2} C \sqrt{1 - k_\theta^C(\mathbf{x}, \mathbf{x}')} \, ,$$

yielding $\|\widehat{y}_i - \widehat{y}_j\| \leqslant \sqrt{2} C \sqrt{1 - k_\theta^C(\mathbf{x}_i, \mathbf{x}_j)}$ and thus $\big| \mathbb{E}_k[\widehat{y}_i - \widehat{y}] \big| \leqslant \sqrt{2} C \, \mathbb{E}_k \Big[\sqrt{1 - k_\theta^C(\mathbf{x}_i, \cdot)} \Big].$

7 Conclusion

We defined a proper notion of input similarity as perceived by the neural network, based on the ability of the network to distinguish the inputs. This brings a new tool to analyze trained networks, in plus of visualization tools such as grad-CAM [23]. We showed how to turn it into a density estimator, which was validated on a controlled experiment, and usable to perform fast statistics on large datasets. It opens the door to underfit/overfit/uncertainty analyses or even control during training, as it is differentiable and computable at low cost. We also showed that any desired similarity could be enforced during training, at reasonable cost, and noticed a dataset-dependent boosting effect that should be further studied along with robustness to adversarial attacks, as such training differs significantly from usual methods. Finally, we extended *noise2noise* [14] to the case of non-identical inputs, thus expressing self-denoising effects as a function of inputs' similarities. The code is available at `https://github.com/Lydorn/netsimilarity` .

Acknowledgments

References

[1] Kyle Bradbury, Benjamin Brigman, Leslie Collins, Timothy Johnson, Sebastian Lin, Richard Newell, Sophia Park, Sunith Suresh, Hoel Wiesner, and Yue Xi. Aerial imagery object identification dataset for building and road detection, and building height estimation, July 2016.

[2] Lenaic Chizat and Francis Bach. A note on lazy training in supervised differentiable programming. *arXiv preprint arXiv:1812.07956*, 2018.

[3] Taco Cohen and Max Welling. Group equivariant convolutional networks. In *International conference on machine learning*, pages 2990–2999, 2016.

[4] Harris Drucker and Yann Le Cun. Double backpropagation increasing generalization performance. In *IJCNN-91-Seattle International Joint Conference on Neural Networks*, volume 2, pages 145–150. IEEE, 1991.

[5] Leon Gatys, Alexander S Ecker, and Matthias Bethge. Texture synthesis using convolutional neural networks. In *Advances in neural information processing systems*, pages 262–270, 2015.

[6] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

[7] Nicolas Girard, Guillaume Charpiat, and Yuliya Tarabalka. Noisy Supervision for Correcting Misaligned Cadaster Maps Without Perfect Ground Truth Data. In *IGARSS*, July 2019. URL `https://hal.inria.fr/hal-02065211`.

[8] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems*, pages 5767–5777, 2017.

[9] Frank R. Hampel. The influence curve and its role in robust estimation. *Journal of the American Statistical Association*, 69(346):383–393, 1974. ISSN 01621459. URL `http://www.jstor.org/stable/2285666`.

[10] Sepp Hochreiter and Jürgen Schmidhuber. Simplifying neural nets by discovering flat minima. In *Advances in neural information processing systems*, pages 529–536, 1995.

[11] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pages 8571–8580, 2018.

[12] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision*, pages 694–711. Springer, 2016.

[13] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *International Conference on Machine Learning*, pages 1885–1894. PMLR, 2017.

[14] Jaakko Lehtinen, Jacob Munkberg, Jon Hasselgren, Samuli Laine, Tero Karras, Miika Aittala, and Timo Aila. Noise2noise: Learning image restoration without clean data. In *International Conference on Machine Learning*, pages 2971–2980, 2018.

[15] Yuncheng Li, Jianchao Yang, Yale Song, Liangliang Cao, Jiebo Luo, and Li-Jia Li. Learning from noisy labels with distillation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1910–1918, 2017.

[16] Emmanuel Maggiori, Yuliya Tarabalka, Guillaume Charpiat, and Pierre Alliez. Can semantic labeling methods generalize to any city? the Inria aerial image labeling benchmark. In *IGARSS*, 2017.

[17] Volodymyr Mnih and Geoffrey E Hinton. Learning to label aerial images from noisy data. In *Proceedings of the 29th International conference on machine learning (ICML-12)*, pages 567–574, 2012.

[18] Nagarajan Natarajan, Inderjit S Dhillon, Pradeep K Ravikumar, and Ambuj Tewari. Learning with noisy labels. In *Advances in neural information processing systems*, pages 1196–1204, 2013.

[19] Yann Ollivier. Riemannian metrics for neural networks I: feedforward networks. *Information and Inference: A Journal of the IMA*, 4(2):108–153, 03 2015. ISSN 2049-8772. doi: 10.1093/imaiai/iav006. URL `https://doi.org/10.1093/imaiai/iav006`.

[20] Yann Ollivier. Riemannian metrics for neural networks II: recurrent networks and learning symbolic data sequences. *Information and Inference: A Journal of the IMA*, 4(2):154–193, 03 2015. ISSN 2049-8772. doi: 10.1093/imaiai/iav007. URL `https://doi.org/10.1093/imaiai/iav007`.

[21] OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org , 2017.

[22] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, pages 833–840. Omnipress, 2011.

[23] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 618–626, 2017.

[24] Sainbayar Sukhbaatar, Joan Bruna, Manohar Paluri, Lubomir Bourdev, and Rob Fergus. Training convolutional networks with noisy labels. *arXiv preprint arXiv:1406.2080*, 2014.

[25] Chih-Kuan Yeh, Joon Kim, Ian En-Hsu Yen, and Pradeep K Ravikumar. Representer point selection for explaining deep neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31, pages 9291–9301. Curran Associates, Inc., 2018. URL `https://proceedings.neurips.cc/paper/2018/file/8a7129b8f3edd95b7d969dfc2c8e9d9d-Paper.pdf`.

A Code

The whole code (image registration, experiments to test density estimators, enforcing similarity...) is available on the following github repository: `https://github.com/Lydorn/netsimilarity` .

B Proofs of the properties of the 1D similarity kernel

We give here the proofs at the properties of the 1-dimensional-output similarity kernel.

B.1 Proof of Theorem 1

Theorem 1. *For any real-valued neural network f_θ whose last layer is a linear layer (without any parameter sharing) or a standard activation function thereof (sigmoid, tanh, ReLU...), and for any inputs $\mathbf{x}$ and $\mathbf{x}'$,*

$$\nabla_\theta f_\theta(\mathbf{x}) = \nabla_\theta f_\theta(\mathbf{x}') \implies f_\theta(\mathbf{x}) = f_\theta(\mathbf{x}') .$$

Proof. If the last layer is linear, the output is of the form $f_\theta(\mathbf{x}) = \sum_i w_i a_i(\mathbf{x}) + b$, where w_i and b are parameters in $\mathbb{R}$ and $a_i(\mathbf{x})$ activities from previous layers. The gradient $\nabla_\theta f_\theta(\mathbf{x})$ contains in particular as coefficients the derivatives $\frac{df_\theta(\mathbf{x})}{dw_i} = a_i(\mathbf{x})$. Thus $\nabla_\theta f_\theta(\mathbf{x}) = \nabla_\theta f_\theta(\mathbf{x}') \implies a_i(\mathbf{x}) = a_i(\mathbf{x}') \; \forall i$ in the last layer. The outputs can be then rebuilt: $f_\theta(\mathbf{x}) = \sum_i w_i a_i(\mathbf{x}) + b = \sum_i w_i a_i(\mathbf{x}') + b = f_\theta(\mathbf{x}')$.

If the output is of the form $f_\theta(\mathbf{x}) = \sigma(c(\mathbf{x}))$ with $c(\mathbf{x}) = \sum_i w_i a_i(\mathbf{x}) + b$, then the gradient equality implies $\frac{df_\theta(\mathbf{x})}{db} = \frac{df_\theta(\mathbf{x}')}{db}$, whose value is $\sigma'(c(\mathbf{x})) = \sigma'(c(\mathbf{x}'))$. Then, as $\sigma'(c(\mathbf{x})) a_i(\mathbf{x}) = \frac{df_\theta(\mathbf{x})}{dw_i} = \frac{df_\theta(\mathbf{x}')}{dw_i} = \sigma'(c(\mathbf{x}')) a_i(\mathbf{x}')$, we can deduce $a_i(\mathbf{x}) = a_i(\mathbf{x}')$ for all i provided $\sigma'(c(\mathbf{x})) \neq 0$. In that case, from these identical activities one can rebuild identical outputs. Otherwise, $\sigma'(c(\mathbf{x})) = \sigma'(c(\mathbf{x}')) = 0$, which is not possible with strictly monotonous activation functions, such as tanh or sigmoid. For ReLU, $\sigma'(c(\mathbf{x})) = 0 \implies \sigma(c(\mathbf{x})) = 0$ and thus $f_\theta(\mathbf{x}) = f_\theta(\mathbf{x}') = 0$. The same reasoning holds for other activation functions with only one flat piece (such as the ReLU negative part), *i.e.* for which the set $\sigma(\sigma'^{-1}(\{0\}))$ is a singleton. $\qquad\square$

B.2 Proof of Corollary 1

Corollary 1. *Under the same assumptions, for any inputs* $\mathbf{x}$ *and* $\mathbf{x}'$,

$$k_\theta^C(\mathbf{x}.\mathbf{x}') = 1 \quad \implies \quad \nabla_\theta f_\theta(\mathbf{x}) = \nabla_\theta f_\theta(\mathbf{x}'),$$
$$\text{hence} \quad k_\theta^C(\mathbf{x}.\mathbf{x}') = 1 \quad \implies \quad f_\theta(\mathbf{x}) = f_\theta(\mathbf{x}').$$

Proof. $k_\theta^C(\mathbf{x}.\mathbf{x}') = 1$ means $\frac{\nabla_\theta f_\theta(\mathbf{x})}{\|\nabla_\theta f_\theta(\mathbf{x})\|} \cdot \frac{\nabla_\theta f_\theta(\mathbf{x}')}{\|\nabla_\theta f_\theta(\mathbf{x}')\|} = 1$, which implies $\exists \alpha \in \mathbb{R}^*, \; \nabla_\theta f_\theta(\mathbf{x}) = \alpha \nabla_\theta f_\theta(\mathbf{x}')$. We need to show that $\alpha = 1$. Under the assumptions of Theorem 1, following its proof:

- either the last layer is linear, the output is of the form $f_\theta(\mathbf{x}) = \sum_i w_i a_i(\mathbf{x}) + b$, and then $\nabla_b f_\theta(\mathbf{x}) = \alpha \nabla_b f_\theta(\mathbf{x}')$ while $\frac{df_\theta(\mathbf{x})}{db} = 1$ and $\frac{df_\theta(\mathbf{x}')}{db} = 1$, hence $\alpha = 1$;

- either the output is of the form $f_\theta(\mathbf{x}) = \sigma(c(\mathbf{x}))$ with $c(\mathbf{x}) = \sum_i w_i a_i(\mathbf{x}) + b$, and then $\sigma'(c(\mathbf{x})) = \nabla_b f_\theta(\mathbf{x}) = \alpha \nabla_b f_\theta(\mathbf{x}') = \alpha \sigma'(c(\mathbf{x}'))$, while, for any i, $\sigma'(c(\mathbf{x})) a_i(\mathbf{x}) = \frac{df_\theta(\mathbf{x})}{dw_i} = \alpha \frac{df_\theta(\mathbf{x}')}{dw_i} = \alpha \sigma'(c(\mathbf{x}')) a_i(\mathbf{x}')$. Thus, supposing $\sigma'(c(\mathbf{x})) \neq 0$, we obtain $a_i(\mathbf{x}) = a_i(\mathbf{x}') \; \forall i$, and thus we can rebuild from the activities $c(\mathbf{x}) = c(\mathbf{x}')$, from which $\sigma'(c(\mathbf{x})) = \sigma'(c(\mathbf{x}'))$ and thus $\alpha = 1$. Otherwise, $\sigma'(c(\mathbf{x})) = \sigma'(c(\mathbf{x}')) = 0$ and the two full gradients $\nabla_\theta f_\theta(\mathbf{x})$ and $\nabla_\theta f_\theta(\mathbf{x}')$ are 0 and thus equal.

The conditions for $k_\theta^C(\mathbf{x}.\mathbf{x}') = 1 \implies \nabla_\theta f_\theta(\mathbf{x}) = \nabla_\theta f_\theta(\mathbf{x}')$ to hold are actually much weaker: it is sufficient that in the whole network architecture there exists *one* useful neuron (in the sense of the next paragraph) of that type (so called *linear* but actually affine).

$\qquad\square$

B.3 Proof of Theorem 2

Theorem 2. *For any real-valued neural network* f_θ *without parameter sharing, if* $\nabla_\theta f_\theta(\mathbf{x}) = \nabla_\theta f_\theta(\mathbf{x}')$ *for two inputs* $\mathbf{x}, \mathbf{x}'$, *then all useful activities computed when processing* $\mathbf{x}$ *are equal to the ones obtained when processing* $\mathbf{x}'$.

We name *useful* activities all activities whose variation would have an impact on the output, *i.e.* all the ones satisfying $\frac{df_\theta(\mathbf{x})}{da_i} \neq 0$. This condition is typically not satisfied when the activity is multiplied by 0, *i.e.* $w_i = 0$, or when it is negative and followed by a ReLU, or when all its contributions to the output annihilate together (*e.g.*, a sum of two neurons with opposite weights: $f_\theta(\mathbf{x}) = \sigma(a_i(\mathbf{x})) - \sigma(a_i(\mathbf{x}))$).

Proof. Let $a_i(\mathbf{x})$ be a useful activity (for $\mathbf{x}$). It is fed to at least one useful neuron, whose pre-activation output is of the form $c(\mathbf{x}) = \sum_i w_i a_i(\mathbf{x}) + b$. Then $\frac{df_\theta(\mathbf{x})}{db} = \frac{df_\theta(\mathbf{x})}{dc} \neq 0$ (the output of the neuron is useful), and $\frac{df_\theta(\mathbf{x})}{dw_i} = \frac{df_\theta(\mathbf{x})}{db} a_i(\mathbf{x})$. From the gradient equality, $a_i(\mathbf{x}) = \frac{df_\theta(\mathbf{x})}{dw_i} / \frac{df_\theta(\mathbf{x})}{db} = \frac{df_\theta(\mathbf{x}')}{dw_i} / \frac{df_\theta(\mathbf{x}')}{db} = a_i(\mathbf{x}')$. $\qquad\square$

13

C Higher output dimension

We expand here all the mathematical aspects of the homonymous section of the article.

C.1 Derivation

Let us now study the case where $f_\theta(\mathbf{x})$ is a vector in $\mathbb{R}^d$ with $d > 1$.

The optimal parameter change $\delta\theta$ to push $f_\theta(\mathbf{x})$ in a direction $\mathbf{v}$ (with a force ε) is less straightforward to obtain. First, one can define as many gradients as output coordinates: $\nabla_\theta f_\theta^i(\mathbf{x})$, for $i \in [\![1, d]\!]$.

This family of gradients can be shown to be linearly independent, unless the architecture of the network is specifically built not to. If for instance each output coordinate has its own bias parameter, *i.e.* writes in the form $f_\theta^i(\mathbf{x}) = b_i + g_\theta(\mathbf{x})$ or $\sigma(b_i + g_\theta(\mathbf{x}))$ with a strictly monotonous activation function σ, then the derivative w.r.t. b_i will be 1 (or σ') only in the i-th gradient and 0 in the other ones. Thus the j-th gradient contains in particular the subvector $(\frac{df^j}{db_i})_i = (\delta_{i=j})_i$, and the gradients are consequently independent. In the case where all coordinates depend on all biases, but not identically, as with a softmax, the argument stays true.

Any parameter variation $\delta\theta \in \mathbb{R}^p$ can then be uniquely decomposed as:

$$\delta\theta = \sum_{i=1}^d \alpha_i \nabla f_\theta^i(\mathbf{x}) \; + \; \gamma$$

where $\alpha_i \in \mathbb{R}$ and where $\gamma \in \mathbb{R}^p$ is orthogonal to all coordinate gradients. This parameter variation induces an output variation:

$$f_{\theta+\delta\theta}(\mathbf{x}) - f_\theta(\mathbf{x}) = \nabla_\theta f_\theta(\mathbf{x})\, \delta\theta + O(\|\delta\theta\|^2)$$

$$= \left(\sum_i \alpha_i \nabla_\theta f_\theta^i(\mathbf{x}) \cdot \nabla f_\theta^j(\mathbf{x})\right)_j + 0 + O(\|\delta\theta\|^2)$$

$$= C\alpha + O(\|\alpha\|^2)$$

where C is the correlation matrix of the gradients: $C_{ij} = \nabla_\theta f_\theta^i(\mathbf{x}) \cdot \nabla f_\theta^j(\mathbf{x})$. It turns out that C is invertible:

$$C\alpha = 0 \implies \alpha C\alpha = 0 \implies \alpha \nabla_\theta f_\theta(\mathbf{x})\, \nabla_\theta f_\theta(\mathbf{x})\, \alpha = 0$$

$$\implies \|\nabla_\theta f_\theta(\mathbf{x})\, \alpha\|^2 = 0 \implies \sum_i \alpha_i \nabla f_\theta^i(\mathbf{x}) = 0$$

$\implies \alpha = 0$ as the $\nabla_\theta f_\theta^i(\mathbf{x})$ are linearly independent. Thus, for a desired output move in the direction $\mathbf{v}$ with amplitude ε, *i.e.* $f_{\theta+\delta\theta}(\mathbf{x}) - f_\theta(\mathbf{x}) = \varepsilon\mathbf{v}$, one can compute the associated linear combination $\alpha = \varepsilon\, C^{-1}\mathbf{v}$ and thus the smallest associated parameter change $\delta\theta = \sum_i \alpha_i \nabla f_\theta^i(\mathbf{x})$.

The output variation induced at any other point $\mathbf{x}'$ by this parameter change is then:

$$f_{\theta+\delta\theta}(\mathbf{x}') - f_\theta(\mathbf{x}') = \left(\nabla_\theta f_\theta^i(\mathbf{x}') \cdot \delta\theta\right)_i + O(\|\delta\theta\|^2)$$

$$= \left(\sum_j \alpha_j \nabla_\theta f_\theta^i(\mathbf{x}') \cdot \nabla_\theta f_\theta^j(\mathbf{x})\right)_i + O(\|\delta\theta\|^2).$$

$$= \varepsilon\, K_\theta(\mathbf{x}', \mathbf{x})\, C_\theta(\mathbf{x})^{-1}\, \mathbf{v} + O(\varepsilon^2) \tag{6}$$

where the $d \times d$ kernel matrix $K_\theta(\mathbf{x}, \mathbf{x}')$ is defined by $K_\theta^{ij}(\mathbf{x}, \mathbf{x}') = \nabla_\theta f_\theta^i(\mathbf{x}) \cdot \nabla_\theta f_\theta^j(\mathbf{x}')$, and where the matrix $C_\theta(\mathbf{x}) = K_\theta(\mathbf{x}, \mathbf{x})$ is the previously defined self-correlation matrix C. Its role is equivalent of the normalization by $\|\nabla_\theta f_\theta(\mathbf{x})\|^2$ in the 1D case, in plus of decorrelating the gradients.

The interpretation of (3) is that if one moves the output for point $\mathbf{x}$ by $\mathbf{v}$, then the output for point $\mathbf{x}'$ will be moved also, by $M\mathbf{v}$, with $M = K_\theta(\mathbf{x}, \mathbf{x}')\, K_\theta(\mathbf{x}, \mathbf{x})^{-1}$. Note that these matrices M or K are only $d \times d$ where d is the output dimension. They are thus generally small and easy to manipulate or inverse.

C.2 Normalized cross-correlation matrix

The normalized version of the kernel (3) is:

$$K_\theta^C(\mathbf{x}, \mathbf{x}') \;=\; C_\theta(\mathbf{x})^{-1/2}\, K_\theta(\mathbf{x}, \mathbf{x}')\, C_\theta(\mathbf{x}')^{-1/2} \tag{7}$$

which is symmetric in the sense that $K_\theta^C(\mathbf{x}', \mathbf{x}) = K_\theta^C(\mathbf{x}, \mathbf{x}')^T$.

A matrix $K_\theta^C(\mathbf{x}, \mathbf{x}')$ with small coefficients means that $\mathbf{x}$ and $\mathbf{x}'$ are relatively independent, from a neural network point of view (moves at $\mathbf{x}$ won't be transferred to $\mathbf{x}'$). On the opposite, the highest possible dependency is $K_\theta^C(\mathbf{x}, \mathbf{x}) = \mathrm{Id}$.

To study properties of this similarity measure, note that $K_\theta^C(\mathbf{x}, \mathbf{x}') = (G_\mathbf{x}^N)^T\, G_{\mathbf{x}'}^N$ with $G_\mathbf{x}^N = G_\mathbf{x}(G_\mathbf{x}^T G_\mathbf{x})^{-1/2}$, where $G_\mathbf{x} = \nabla_\theta f(\mathbf{x})$: it is the product of normalized, decorrelated versions of the gradient. Indeed, at any point $\mathbf{x}$, the normalized gradient matrix $G_\mathbf{x}^N$ satisfies: $(G_\mathbf{x}^N)^T\, G_\mathbf{x}^N = K_\theta^C(\mathbf{x}, \mathbf{x}) = K_\theta(\mathbf{x}, \mathbf{x})^{-1/2} K_\theta(\mathbf{x}, \mathbf{x}) K_\theta(\mathbf{x}, \mathbf{x})^{-1/2} = \mathrm{Id}$ and consequently $G_\mathbf{x}^N$ can be seen as an orthonormal family of vectors $G_\mathbf{x}^{N,i}$.

The L^2 (Frobenius) norm of the ortho-normalized gradient $G_\mathbf{x}^N$ is thus:

$$\left\| G_\mathbf{x}^N \right\|_F^2 = \mathrm{Tr}((G_\mathbf{x}^N)^T\, G_\mathbf{x}^N) = \mathrm{Tr}(\mathrm{Id}) = d \,.$$

At point $\mathbf{x}'$, $G_{\mathbf{x}'}^N$ is also an orthonormal family, but possibly arranged differently or generating a different subspace of $\mathbb{R}^p$. If $G_\mathbf{x}^N$ and $G_{\mathbf{x}'}^N$ generate the same subspace, then their product $(G_\mathbf{x}^N)^T\, G_{\mathbf{x}'}^N$ is an orthogonal matrix Q (change of basis) and its L^2 (Frobenius) norm is then $\left\| Q \right\|_F^2 = \mathrm{Tr}(Q^T Q) = \mathrm{Tr}(\mathrm{Id}) = d$. Otherwise, $(G_\mathbf{x}^N)^T\, G_{\mathbf{x}'}^N$ can be seen as a projection from one subspace to another one, each vector $G_{\mathbf{x}'}^{N,j}$ is projected onto the ortho-normal family $(G_\mathbf{x}^{N,i})_i$, and as a projection decreases the Euclidean norm, $\sum_i \left(G_\mathbf{x}^{N,i} \cdot G_{\mathbf{x}'}^{N,j} \right)^2 \leqslant \left\| G_{\mathbf{x}'}^{N,j} \right\|^2 = 1$. Thus:

$$\left\| K_\theta^C(\mathbf{x}, \mathbf{x}') \right\|_F = \sqrt{\sum_{ij} \left(G_\mathbf{x}^{N,i} \cdot G_{\mathbf{x}'}^{N,j} \right)^2} \leqslant \sqrt{d} \,.$$

Moreover, any coefficient of the kernel matrix satisfies:

$$\left| K_\theta^{C,ij}(\mathbf{x}, \mathbf{x}') \right| = \left| G_\mathbf{x}^{N,i} \cdot G_{\mathbf{x}'}^{N,j} \right| \leqslant \left\| G_\mathbf{x}^{N,i} \right\|_2 \left\| G_{\mathbf{x}'}^{N,j} \right\|_2 = 1$$

as each vector $G_\mathbf{x}^{N,i}$ is unit-norm. This implies in particular that the trace is bounded:

$$-d \;\leqslant\; \mathrm{Tr}(K_\theta^C(\mathbf{x}, \mathbf{x}')) \;\leqslant d.$$

To sum up, the similarity matrix $K_\theta^C(\mathbf{x}, \mathbf{x}')$ satisfies the following properties:

- its coefficients are bounded, in $[-1, 1]$
- its trace is at most d
- its (Frobenius) norm is at most $\sqrt{d}$
- self-similarity is identity: $\forall \mathbf{x}, \;\; K_\theta^C(\mathbf{x}, \mathbf{x}) = \mathrm{Id}$
- the kernel is symmetric, in the sense that $K_\theta^C(\mathbf{x}', \mathbf{x}) = K_\theta^C(\mathbf{x}, \mathbf{x}')^T$.

C.3 Similarity in a single value

Note that when the trace is close to its maximal value d, the diagonal coefficients are close to 1, and their contribution to the Frobenius norm squared is close to d. Therefore, all non-diagonal coefficients are close to 0, and the matrix is close to Id. And reciprocally, a matrix close to Id has a trace close to d. Thus, two related ways to quantify similarity in a single real value in $[-1, 1]$ appear:

- the distance to the identity $D = \left\| K_\theta^C(\mathbf{x}, \mathbf{x}') - \mathrm{Id} \right\|_F$, which can be turned into a similarity as $1 - \frac{1}{\sqrt{d}} D$ or $1 - \frac{1}{2d} D^2$, since $D \in [0, 2\sqrt{d}]$
- the normalized trace: $\frac{1}{d} \mathrm{Tr}\, K_\theta^C(\mathbf{x}, \mathbf{x}')$, which is also the alignment with the identity: $\frac{1}{d} K_\theta^C(\mathbf{x}, \mathbf{x}') \cdot_F \mathrm{Id}$, where $\cdot_F$ denotes the Frobenius inner product (*i.e.* coefficient by coefficient).

The link between these two quantities can be made explicit by developing:

$$\left\| K_\theta^C(\mathbf{x}, \mathbf{x}') - \mathrm{Id} \right\|_F^2 = \left\| K_\theta^C(\mathbf{x}, \mathbf{x}') \right\|_F^2 - 2\mathrm{Tr}(K_\theta^C(\mathbf{x}, \mathbf{x}')) + d$$

which rewrites as:

$$\left(1 - \frac{D^2}{2d} \right) = \frac{\mathrm{Tr}(K_\theta^C(\mathbf{x}, \mathbf{x}'))}{d} + \frac{1}{2} \left(1 - \frac{\left\| K_\theta^C(\mathbf{x}, \mathbf{x}') \right\|_F^2}{d} \right).$$

The last term lies in $[0, 1]$ and measures the mismatch between the vector subspaces generated by the two families of gradients $\left(\nabla_\theta f^i(\mathbf{x}) \right)_i$ and $\left(\nabla_\theta f^i(\mathbf{x}') \right)_i$. It is 1 when $f_\theta(\mathbf{x})$ and $f_\theta(\mathbf{x}')$ can be moved independently, and 0 when they move jointly (though not necessarily in the same direction).

As our two similarity measures $1 - \frac{D^2}{2d}$ and $\frac{1}{d}\mathrm{Tr}(K_\theta^C(\mathbf{x}, \mathbf{x}'))$ have same optimum (Id) and are closely related, in the sequel we will focus on the second one and define:

$$k_\theta^C(\mathbf{x}, \mathbf{x}') \;=\; \frac{1}{d} \, \mathrm{Tr} \, K_\theta^C(\mathbf{x}, \mathbf{x}') . \tag{8}$$

C.4 Metrics on output: rotation-invariance

Similarity in $\mathbb{R}^d$, to compare $\mathbf{v}$ and $\mathbf{v}' = M\mathbf{v}$, might be richer than just checking whether the vectors are equal or close in L^2 norm.

For instance, one could quotient the output space by the group of rotations, in order to express a known or desired equivariance of the network to rotations. If the output is the predicted motion of some object described in the input, one could wish indeed that if the input object is rotated by an angle ϕ, then the output should be rotated as well with the same angle.

In that case, given two inputs $\mathbf{x}$ and $\mathbf{x}'$ and associated output variations $\mathbf{v}$ and $\mathbf{v}'$, without knowing the rotation angle if applicable, one could consider all possible rotated versions $R_\phi \mathbf{v}' = R_\phi M\mathbf{v}$, where R_ϕ is the rotation matrix with angle ϕ, and pick the best angle ϕ that maximizes the alignment $\mathbf{v} \cdot R_\phi M\mathbf{v}$, *i.e.* such that $R_\phi M$ is the closest to the $d \times d$ identity matrix. This can be computed easily in closed form, for instance in the 2-dimensional case as follows.

The 2×2 matrix of interest (Eq. 4) can be written as the product of two $p \times 2$ matrices of the form $G(G^T G)^{-1/2}$, where G is the matrix containing the gradient of all coordinates. Rotating the coordinates of G amounts to considering $GR_\phi(R_\phi^T G^T G R_\phi)^{-1/2} = G(G^T G)^{-1/2} R_\phi$ instead. Thus the effect of rotation is just right-multiplying our 2×2 matrix M of interest (Eq. 4) by R_ϕ. We are thus interested into getting MR_ϕ as close as possible to the 2×2 identity. For our trace-based similarity kernel (Eq. 5), this amounts to maximizing $\mathrm{Tr}(MR_\phi) = \cos(\phi)(M_{11} + M_{22}) + \sin(\phi)(M_{12} - M_{21})$ w.r.t. ϕ, whose optimal value is:

$$k_\theta^{C,\mathrm{rot}}(\mathbf{x}, \mathbf{x}') = \frac{1}{2} \sqrt{(M_{11} + M_{22})^2 + (M_{12} - M_{21})^2}$$

$$= \frac{1}{2} \sqrt{\left\| M \right\|_F^2 + 2 \det M}$$

where $M = K_\theta^C(\mathbf{x}, \mathbf{x}')$. This quantity is indeed rotation-invariant, as the Frobenius norm and the determinant do not change upon rotations. Note that one could also consider instead the subspace match $\frac{1}{d}\left\| M \right\|_F^2$. The main difference between the two is that the first one penalizes mirror symmetries (through $\det M$) while the second one does not.

Note that other metrics are possible in the output space. For instance, the loss metric quantifies the norm of a move $\mathbf{v}$ by its impact on the loss $\left. \frac{dL(y)}{dy} \right|_{f_\theta(\mathbf{x})}(\mathbf{v})$. It has a particular meaning though, and is relevant only if well designed and not noisy, as seen in the remote sensing image registration example. Note also that in such a case the associated similarity would not be intrinsic anymore to the neural network as it depends on the loss.
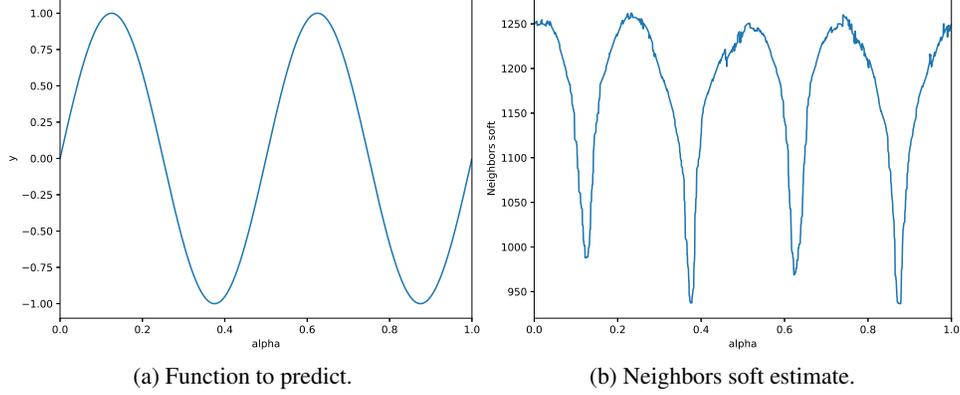
(a) Function to predict.   (b) Neighbors soft estimate.

Figure 9: Toy problem with the frequency f = 2.

D Estimating density

D.1 Toy problem

The toy problem used in the paper to test the various estimators for neighbor count estimation consists of predicting a one dimensional function, namely a sinusoid (such as in Fig.9 (a)). We can easily change the difficulty of the problem by using different values of frequency. The neural network would perform this mapping: $y = \sin(2\pi f x), x \in [0, 1]$.

A problem arises however when estimating the number of neighbors because the input space has 2 boundaries at $x = 0$ and $x = 1$, leading to fewer neighbors when x approaches either of those boundaries. To avoid this problem, we transform the input space to a 2D circle. Namely, the task is now $y = sin(2\pi f \alpha(x)), x \in \{(\cos(2\pi\alpha), \sin(2\pi\alpha)), \alpha \in [0, 1]\}$, with the input space having no boundaries.

The dataset is generated with n=2048 input points. The network used is fully-connected and has 5 hidden layers of 64 neurons trained with the Adam optimizer for 80 epochs with a base learning rate of $1e^{-4}$. An experiment consist of training the network on a dataset generated with a specific frequency f. Each experiment was repeated 5 times, in order to take the median of every result to limit the variance due to the neural network stochastic training.

We can see in Fig.9 (b) the proposed soft estimate k_θ^C for each input point (projected to 1D). As expected we observe that the number of neighbors drops when the curvature is high: the objective changes quickly and the network adjusts to better distinguish inputs in places of higher curvature.

D.2 Other possible uses

Density homogeneity as an optimization criterion   The estimations above are meant to be done post-training. This said, one could control density explicitly, by computing the number of neighbors for all points, and asking it to be in a reasonable range, or in a reasonable proportion q of the dataset size $\mathcal{D}$, by adding *e.g.* to the loss $\sum_i \left(\frac{N_S(\mathbf{x}_i)}{\mathcal{D}} - q \right)^2$. Online learning could also make use of such tools, to sample first lowly-populated areas, where uncertainty is higher.

E Enforcing similarity

We give here a few more details on the homonymous section of the paper.

E.1 Complexity

A gradient descent step on this quantity for a given pair $(\mathbf{x}, \mathbf{x}')$ (in a mini-batch approach, *e.g.*) requires the computation of the gradient $\nabla_\theta k_\theta^C(\mathbf{x}, \mathbf{x}') = \nabla_\theta (\nabla_\theta f_\theta(\mathbf{x}) \cdot \nabla_\theta f_\theta(\mathbf{x}'))$. While a naive approach would require the computation of a second derivative, *i.e.* a matrix of size $p \times p$ where p is
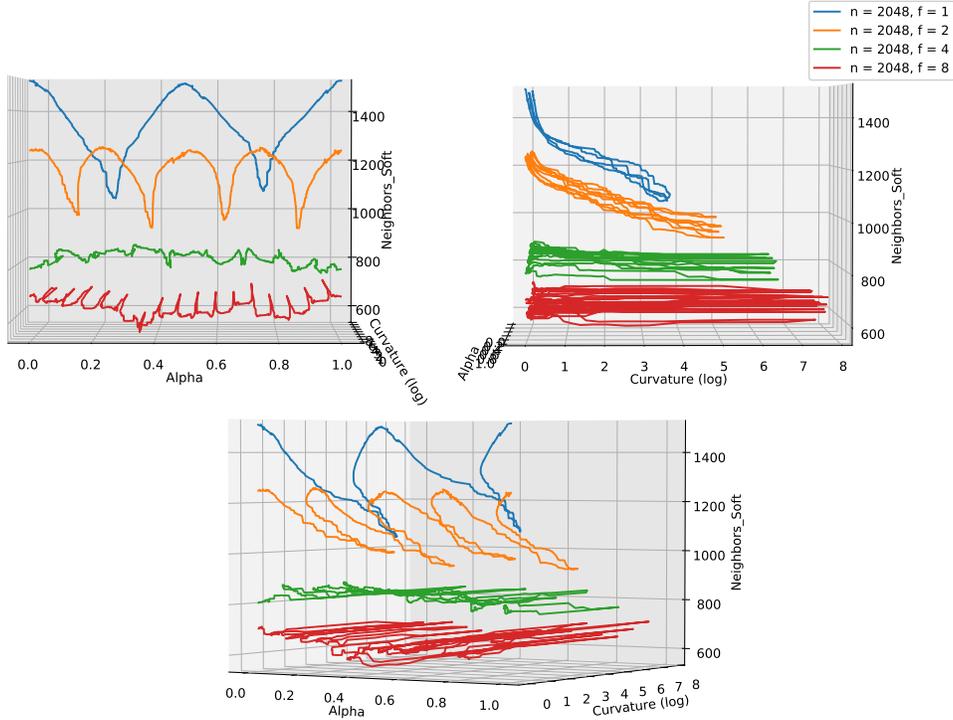
Figure 10: 3D plot of neighbors soft with varying frequency. Script and data to plot interactively in attached files. Run the bash script "main_plot_exps.paper.sh" to reproduce this exact figure. Alternatively use "main_plot_exps.py" with arguments of your choosing to plot different values (run "python main_plot_exps.py -h" to see possible arguments).

the number of parameters, it is actually possible to compute $\nabla_\theta k_\theta^C(\mathbf{x}, \mathbf{x}') = \nabla_\theta \sum_i \frac{df_\theta(\mathbf{x})}{d\theta_i} \frac{df_\theta(\mathbf{x}')}{d\theta_i}$ in linear time $O(p)$, taking advantage of the serial structure of the computational graph. The framework enabling such computations is already available on common deep learning platforms, initially intended for the computation of $\nabla_\mathbf{x} \nabla_\theta f_\theta(\mathbf{x})$ for some variations on GANs.

E.2 Group invariance

Dataset augmentation is a standard machine learning technique; when augmenting the dataset by a group transformation of the input (*e.g.*, translation, rotation...) or by small intensity noise, new samples are artificially created, to augment the dataset size and hope for invariance to such transformations. One can ask the network to consider orbits of samples as similar with the technique above.

Furthermore, if the group infinitesimal elements are expressible as differential operators e_k, one could require directly, for all $\mathbf{x}$, invariance in the tangent plane in the directions of these differential operators:

$$\|\partial_\mathbf{x} \nabla_\theta f(\mathbf{x}) \cdot e_k(\mathbf{x})\|^2$$

which is the limit of $\frac{1}{\varepsilon^2} \|\nabla_\theta f_\theta(\mathbf{x}) - \nabla_\theta f_\theta(\mathbf{x} + \varepsilon e_k(\mathbf{x}))\|^2$ when $\varepsilon \to 0$. For instance, in the case of image translations, the operator is $e : \mathbf{x} \mapsto \nabla_x \mathbf{x}(x)$ where x denotes spatial coordinates in the image $\mathbf{x}$, as $\mathbf{x}(x + \tau) = \mathbf{x}(x) + \tau \cdot \nabla_x \mathbf{x}(x) + O(\tau^2)$. This is however not recommended, as representing a translation with such a spatially-local operator does not take into account the spatially-irregular nature of image intensities.

Note that to the opposite of standard robustification techniques considering regularizers such as $\sum_\mathbf{x} \|\nabla_\theta f_\theta(\mathbf{x})\|^2$, we ask not gradients to be always small, but to be smooth, and in certain directions only.

18

E.3 Dynamics of learning: Experimentation details

The results in figure 6 show the average and standard deviation over 60 runs for each curve. The x-axis is the number of batches to the network is trained on (with a batch size of 16). The y-axis is the accuracy metric on the whole validation set. The network architecture is made of 2 convolutions layers (with a kernel size of 5), 2 linear layers and uses PReLU non-linearities. We used Adam with a learning rate of 1e-3 and no weight decay.

We tested other architecures on MNIST: one with residual blocks, one deeper (8 convolutions) and one with tanh non-linearities. Similar results were observed on all cases. Additional tests were performed on CIFAR10 with a VGG architecture and only negligible benefits were observed.

F Noisy Map Alignment Analysis

The task here it to align maps in the form of a list of polygons with remote sensing images while using only the available noisy annotations. We analyze the model developed in a previous work [7]. Specifically, the model is trained in a multiple-rounds training scheme to iteratively align the available noisy annotations, which provides a better ground truth used to train a better model in the next round. An open question is why multiple rounds are needed in this noisy supervision setting, and why not all the noise can be removed in a single training step.

More specifically, the model is made out of 4 neural networks. Each is trained on a different resolution (in terms of ground pixel size) and are applied in a multi-resolution pyramidal manner. In all our experiments we only analyzed the networks trained for a ground pixel size of 4 time smaller than the reference ground pixel size which is $0.3m$. We used the already-trained networks for each round, of which there are 3.

The network was trained with small patches of (image, misaligned map) pairs from images of the Inria dataset [16] and the Bradbury dataset [1]. Ideally we would want to compute the similarities of every possible pairs of inputs, with a small patch size of 124 px. However, given that a typical image of the training dataset is 1250×1250 px (after rescaling) and there are a few hundred of them (328 from the Inria dataset, only counting images where OSM annotations [21]), this would result in 32800 patches. The resulting amount of similarities to compute would be around half a billion. As the network has a few million of parameters and the output is 2D, each computation of similarity takes around 0.5s. To make any computation feasible, we first sample 10 patches per image from the 328 of the Inria dataset. Those patches are chosen at random, as long as there is at least one building lying fully in the patch. As some images have rather sparse buildings, some images give less than 10 patches. We thus obtain 3045 patches representing the dataset. The amount of similarities to compute would be close to 5 million. To study all patches globally, we can use the soft neighbors estimator k_θ^C which has a linear complexity and allows us to compute the amount of neighbors for all 3045 patches in under an hour. However it is also interesting to go in deeper detail and compute similarities for some input pairs. We thus furthermore reduce the amount of pairs by estimating all similarities only for a very small number of patches, for example 10. This results in a 10×3045 similarity matrix.

F.1 Soft estimate on a sampling of the training dataset

In this section we present the results of computing the soft neighbors estimator k_θ^C on the 3045 sampled patches of inputs. We obtain results for the 3 networks of the 3 rounds of the noisy-supervision multi-rounds training scheme. Fig.11 shows a histogram of the soft neighbors estimations. It additionally representative input patches for each bin of the histogram. Those representative patches are chosen so that their neighbor count is closest to the right edge of that bin. We especially observe that inputs in round 2 have more neighbors than the other 2 rounds. This particularity of round 2 will be seen throughout the remaining results. It is the round that aligns the most the annotations (see the Fig.2 on accuracy cumulative distributions in the paper). Round 3 does not perform any more alignment, that might be the reason why its results are different from those of round 2.

F.2 Similarities on pairs of input patches

In this section are the results for the computation of similarities between pairs of input patches. In a first experiment, for every round we chose the 10 patches shown in Fig.11, and computed their

similarities with all the other 3045 patches. In order to visualize this data, we computed the 10-nearest neighbors in terms of similarity for each of those patches, see Fig.12, 13, 14. We computed the histogram of similarities as well, see Fig.15.

In a second experiment, to better compare between rounds, we used another set of 10 patches, this time the same set for each round. Specifically, we sampled 10 patches from the bloomington22 image of the Inria dataset. As just before we computed the 10-nearest neighbors (Fig.16, 17, 18) and the histogram of similarities(Fig.19) for a visualization of those measures.

Generally speaking, inputs in round 2 have more neighbors and the 10-nearest ones are closer than in other rounds (see Fig.12, 13, 14 and Fig.16, 17, 18). For each parch, its closest neighbors generally (for similarity > 0.8) look similar from a human point of view. For example patches with sparse houses and trees have the same kind of neighbors. The same can be said for patches with parking lots and big roads. Another group are patches that are almost empty of buildings, with a lot of low vegetation. Other patch nearest neighbors are more difficult to interpret. In Fig.15 and Fig.19 we can see that for round 2, the spread of the similarities of the selected patches is smaller and the peak of the histogram are closer to the right, meaning all patches are closer than in other rounds. Additionally in Fig.15 we can observe that the bottom patch has closer neighbors than the top patch, this is because the top patch corresponds to the left patch in 11 and the bottom one corresponds to the right patch in 11.

G   Proof details of the self-denoising effect quantification

G.1   Magnitude of kernel-smoothed i.i.d. noise

We show here that $\mathbb{E}_k[\varepsilon] \propto \text{var}_\varepsilon(\mathbb{E}_k[\varepsilon])^{1/2} = \sigma_\varepsilon \, \|k_\theta^{IN}\|_{L2}$.

Let us denote by $\mathbb{E}_\varepsilon[\,]$ and $\text{var}_\varepsilon(\,)$ the expectation and variance with respect to the random variable ε. As a reminder, by assumptions in the noise definition, $\varepsilon = (\varepsilon_i)_i$ is a random, i.i.d. noise, centered and of variance σ_ε.

This is not to be confused with the symbol $\mathbb{E}_k[\,]$, which was defined as, for any vector field a:

$$\mathbb{E}_k[a] = \sum_j a_j \, k_\theta^{IN}(\mathbf{x}_j, \mathbf{x}_i) \,,$$

i.e. as the mean value of a in the neighborhood of i, that is, the weighted average of the a_j with weights $k_\theta^{IN}(\mathbf{x}_j, \mathbf{x}_i)$, which are positive and sum up to 1.

Given a network and its associated kernel k_θ^{IN}, we are interested in to knowing the typical values of $\mathbb{E}_k[\varepsilon]$ for random ε. First, the expectation over the noise of $\mathbb{E}_k[\varepsilon]$ is:

$$\mathbb{E}_\varepsilon\left[\mathbb{E}_k[\varepsilon]\right] \;=\; \mathbb{E}_\varepsilon\left[\sum_j \varepsilon_j \, k_\theta^{IN}(\mathbf{x}_j, \mathbf{x}_i)\right] \;=\; \sum_j \mathbb{E}_\varepsilon[\varepsilon_j] \, k_\theta^{IN}(\mathbf{x}_j, \mathbf{x}_i) \;=\; 0$$

as ε is a centered noise. Thus the random variable $\mathbb{E}_k[\varepsilon]$ is also centered, and therefore its typical values are described by its standard deviation, which is the square root of its variance:

$$\mathbb{E}_k[\varepsilon] \;\propto\; \text{var}_\varepsilon\left(\mathbb{E}_k[\varepsilon]\right)^{1/2} \;.$$

20

The variance can be computed as follows:

$$
\begin{aligned}
\operatorname*{var}_{\varepsilon}\left(\mathbb{E}_{k}[\varepsilon]\right) &= \mathbb{E}_{\varepsilon}\left[\left(\sum_{j}\varepsilon_{j}\,k_{\theta}^{IN}(\mathbf{x}_{j},\mathbf{x}_{i})\right)^{2}\right] \\
&= \mathbb{E}_{\varepsilon}\left[\sum_{j}\varepsilon_{j}^{2}\left(k_{\theta}^{IN}(\mathbf{x}_{j},\mathbf{x}_{i})\right)^{2}\right] \quad \text{as } \varepsilon \text{ is i.i.d.} \\
&= \sigma_{\varepsilon}^{2}\sum_{j}\left(k_{\theta}^{IN}(\mathbf{x}_{j},\mathbf{x}_{i})\right)^{2} \\
&= \sigma_{\varepsilon}^{2}\left\|k_{\theta}^{IN}(\cdot,\mathbf{x}_{i})\right\|_{L2}^{2}.
\end{aligned}
$$

As the weights $p_{j} = k_{\theta}^{IN}(\mathbf{x}_{j},\mathbf{x}_{i})$, for given i and varying j, are positive and sum up to 1, they form a probability distribution. Hence the value of $\left\|k_{\theta}^{IN}(\cdot,\mathbf{x}_{i})\right\|_{L2}^{2} = \|p\|_{L2}^{2}$ satisfies:

- $\|p\|_{L2} \leqslant 1$, as $\sum_{j}p_{j}^{2} \leqslant \sum_{j}p_{j} = 1$, with equality only when $p_{j} = p_{j}^{2}\ \forall j$, that is, all $p_{j} = 0$ except for one $p_{j*} = 1$, which means $k_{\theta}^{I}(\mathbf{x}_{j},\mathbf{x}_{i}) = 0 \quad \forall j \neq i$, which means that all data samples are fully independent from the network's point of view.

- $\|p\|_{L2} \geqslant \frac{1}{\sqrt{N}}$ as $1 = \sum_{j}1 \times p_{j} \leqslant \|1\|_{L2}\|p\|_{L2} = \sqrt{N}\|p\|_{L2}$ (Cauchy-Bunyakovsky-Schwarz), with equality reached for the uniform distribution: $p_{j} = \frac{1}{N}\ \forall j$, where N is the number of data samples. This implies that all $k_{\theta}^{C}(\mathbf{x}_{j},\mathbf{x}_{i})$ are equal, for all i,j, hence they are all equal to $k_{\theta}^{C}(\mathbf{x}_{i},\mathbf{x}_{i}) = 1$. This is the case studied in [14]: all input points are identical.

The denoising factor $\|k_{\theta}^{IN}(\cdot,\mathbf{x}_{i})\|_{L2}$, which depends on the data point $\mathbf{x}_{i}$ considered, thus expresses where the neighborhood of $\mathbf{x}_{i}$ lies, between these two extremes (all $\mathbf{x}_{j}$ very different from $\mathbf{x}_{i}$, or all identical).

Note: the results above remain valid when the output is higher-dimensional, under the supplementary assumption that the covariance matrix of the noise is proportional to the Identity matrix (*i.e.*, the noises on the various coefficients of the label vector are independent from each other, and follow the same law, with standard deviation σ_{ε}). If not, the expression for $\operatorname{covar}_{\varepsilon}(\mathbb{E}_{k}[\varepsilon])$ is more complex, as Σ_{ε} and k_{θ}^{IN} interact. Note that when the output is of dimension d, the kernel $k_{\theta}^{IN}(\mathbf{x}_{j},\mathbf{x}_{i})$ is a $d \times d$ matrix, thus the denoising factor $\left\|k_{\theta}^{IN}(\cdot,\mathbf{x}_{i})\right\|_{L2}^{2}$ has to be replaced with the matrix $\sum_{j}k_{\theta}^{IN}(\mathbf{x}_{j},\mathbf{x}_{i})\,k_{\theta}^{IN}(\mathbf{x}_{j},\mathbf{x}_{i})^{T}$, which can be summarized by its trace, which is the L^{2} norm of the Frobenius norms: $\left\|\left\|k_{\theta}^{IN}(\cdot,\mathbf{x}_{i})\right\|_{F}\right\|_{L2}^{2}$.

G.2 The function: gradient $\mapsto$ output is Lipschitz

Theorem 1 implies that the application: $\frac{\nabla_{\theta}f_{\theta}(\mathbf{x})}{\|\nabla_{\theta}f_{\theta}(\mathbf{x})\|} \mapsto f_{\theta}(\mathbf{x})$ is well-defined. We show here that this application is also Lipschitz, with a network-dependent constant, under mild hypotheses.

We consider the same assumptions as in Theorem 1 : f_{θ} is a real-valued network, whose last layer is a linear layer or a standard activation function thereof (such as sigmoid, tanh, ReLU...), without parameter sharing (in that last layer). We will also require that the derivative of the activation function is bounded, which is a safe assumption for all networks meant to be trained by gradient descent. Another, technical property (bounded input space) will be assumed in order to imply bounded gradients. A side note indicates how to rewrite the desired property if the input space is not bounded.

Let $\mathbf{x}$ and $\mathbf{x}'$ be any two inputs. We want to bound $|f_{\theta}(\mathbf{x}) - f_{\theta}(\mathbf{x}')|$ by $\|\mathbf{u} - \mathbf{u}'\|_{2}$ times some constant, where $\mathbf{u} = \frac{\nabla_{\theta}f_{\theta}(\mathbf{x})}{\|\nabla_{\theta}f_{\theta}(\mathbf{x})\|}$ and $\mathbf{u}' = \frac{\nabla_{\theta}f_{\theta}(\mathbf{x}')}{\|\nabla_{\theta}f_{\theta}(\mathbf{x}')\|}$.

Let us denote the non-normalized gradients by $\mathbf{v} = \nabla_{\theta}f_{\theta}(\mathbf{x})$ and $\mathbf{v}' = \nabla_{\theta}f_{\theta}(\mathbf{x}')$. We have $\mathbf{u} = \frac{\mathbf{v}}{\|\mathbf{v}\|}$ and $\mathbf{u}' = \frac{\mathbf{v}'}{\|\mathbf{v}'\|}$.

We will proceed in two steps: bounding $|f_\theta(\mathbf{x}) - f_\theta(\mathbf{x}')|$ by $\|\mathbf{v} - \mathbf{v}'\|_2$, and then $\|\mathbf{v} - \mathbf{v}'\|_2$ by $\|\mathbf{u} - \mathbf{u}'\|_2$. The first step is easy and actually sufficient to bound with a non-normalized similarity kernel $k_\theta = \mathbf{v} \cdot \mathbf{v}'$ the shift from the average prediction in the neighborhood. The second step provides a more elegant bound, in that it makes use of the normalized similarity kernel $k_\theta^C = \mathbf{u} \cdot \mathbf{u}'$, but that bound is a priori not as tight and requires more assumptions.

Case where the last layer is linear

The output of the network is of the form

$$f_\theta(\mathbf{x}) = \sum_i w_i a_i(\mathbf{x}) + b \,,$$

where w_i and b are parameters in $\mathbb{R}$ and $a_i(\mathbf{x})$ activities from previous layers. Thus:

$$\begin{aligned}
|f_\theta(\mathbf{x}) - f_\theta(\mathbf{x}')| &= \left| \sum_i w_i \left(a_i(\mathbf{x}) - a_i(\mathbf{x}') \right) \right| \\
&\leqslant \|\mathbf{w}\|_2 \, \|\mathbf{a}(\mathbf{x}) - \mathbf{a}(\mathbf{x}')\|_2 \\
&\leqslant \|\mathbf{w}\|_2 \sqrt{\sum_i (v_i - v_i')^2}
\end{aligned}$$

where the sum is taken over parameters i in the last layer only, using the fact that activities a_i in the last layer are equal to some of the coefficients of the gradient: $v_i := \frac{\partial f_\theta(\mathbf{x})}{\partial w_i} = a_i(\mathbf{x})$.

Note that the derivative with respect to the shift b is $v_b := \frac{\partial f_\theta(\mathbf{x})}{\partial b} = 1$, which ensures that the norm of $\mathbf{v}$ is at least 1. This implies:

$$\|\mathbf{u} - \mathbf{u}'\|_2 \geqslant |u_b - u_b'| = \left| \frac{1}{\|\mathbf{v}\|} - \frac{1}{\|\mathbf{v}'\|} \right|$$

which, combined with:

$$|v_i - v_i'| = \|\mathbf{v}'\| \left| \frac{1}{\|\mathbf{v}'\|} v_i - \frac{v_i'}{\|\mathbf{v}'\|} \right| = \|\mathbf{v}'\| \left| u_i - u_i' + \left(\frac{1}{\|\mathbf{v}'\|} - \frac{1}{\|\mathbf{v}\|} \right) v_i \right|$$

yields:

$$|v_i - v_i'| \leqslant \|\mathbf{v}'\| \left(|u_i - u_i'| + \|\mathbf{u} - \mathbf{u}'\|_2 |v_i| \right) \leqslant \|\mathbf{v}'\| \|\mathbf{u} - \mathbf{u}'\|_2 (1 + |v_i|)$$

from which we finally obtain:

$$|f_\theta(\mathbf{x}) - f_\theta(\mathbf{x}')| \leqslant \left[\|\mathbf{w}\|_2 \|\mathbf{v}'\| \sqrt{\sum_i (1 + |v_i|)^2} \right] \|\mathbf{u} - \mathbf{u}'\|_2$$

which is the bound we were searching for. For the term between brackets to be bounded by a network-dependent constant, one can suppose for instance that the derivative of the activation functions is bounded (which is usually the case for networks meant to be trained by gradient descent), and that the input space is bounded as well; in such cases indeed all coefficients of the gradient vector $\mathbf{v}$ or $\mathbf{v}'$ are bounded, as derivatives of a function composed of constant linear applications (except for the first layer which is a linear application whose factors are bounded inputs, when seen as an application defined on parameters) and of bounded-derivatives activation functions.

Note for unbounded input spaces: If the input space is not bounded, the gradients are not bounded absolutely, as for instance the gradient with respect to a weight in the first layer is the input itself (times a chain product). In that case the application $\mathbf{x} \mapsto \mathbf{v}$ still satisfy a bound of the form $\|\mathbf{v}\| \leqslant (1 + \|\mathbf{x}\|) A$, with A a network-dependent constant (product of determiners of layer weight matrices and of the bound on activation function derivatives to the power: network depth), and thus the application $\mathbf{u} \mapsto f_\theta(\mathbf{x})$ still satisfies a bound of the form, for any $\mathbf{x}, \mathbf{x}'$:

$$|f_\theta(\mathbf{x}) - f_\theta(\mathbf{x}')| \leqslant B (1 + \|\mathbf{x}\|) (1 + \|\mathbf{x}'\|) \|\mathbf{u} - \mathbf{u}'\|_2 \,.$$

The last statement in the paper then becomes

$$\left| \underset{k}{\mathbb{E}}[\widehat{y}_i - \widehat{y}] \right| \;\leqslant\; \sqrt{2}\, B\, (1 + \|\mathbf{x}_i\|)\, \max_j (1 + \|\mathbf{x}_j\|)\, \underset{k}{\mathbb{E}}\left[\sqrt{1 - k_\theta^C(\mathbf{x}_i, \cdot)} \right]$$

which in practice rewrites as the original formulation:

$$\left| \underset{k}{\mathbb{E}}[\widehat{y}_i - \widehat{y}] \right| \;\leqslant\; \sqrt{2}\, C\, \underset{k}{\mathbb{E}}\left[\sqrt{1 - k_\theta^C(\mathbf{x}_i, \cdot)} \right]$$

by taking $C = B \max_j (1 + \|\mathbf{x}_j\|)^2$, considering the actual diameter of the given dataset.

Case where the last layer is an activation function of a linear layer

The output of the network is of the form

$$f_\theta(\mathbf{x}) = \sigma\left(\sum_i w_i a_i(\mathbf{x}) + b \right) \;,$$

and, as the derivative of σ is assumed to be bounded, and as the weights w_i are fixed, $f_\theta(\mathbf{x})$ is a Lipschitz function of the last layer activities $a_i(\mathbf{x})$. Therefore:

$$|f_\theta(\mathbf{x}) - f_\theta(\mathbf{x}')| \leqslant K \|\mathbf{a}(\mathbf{x}) - \mathbf{a}(\mathbf{x}')\|_2 \;.$$

We will denote by α and α' the derivatives with respect to the shift b, which are this time:

$$\alpha := \mathbf{v}_b := \frac{\partial f_\theta(\mathbf{x})}{\partial b} = \sigma'\bigg|_{\sum_i w_i a_i(\mathbf{x})+b} \qquad \text{and} \qquad \alpha' := \mathbf{v}_b' := \frac{\partial f_\theta(\mathbf{x}')}{\partial b} = \sigma'\bigg|_{\sum_i w_i a_i(\mathbf{x}')+b} \;.$$

We proceed as previously:

$$\|\mathbf{u} - \mathbf{u}'\|_2 \;\geqslant\; |u_b - u_b'| \;=\; \left| \frac{\alpha}{\|\mathbf{v}\|} - \frac{\alpha'}{\|\mathbf{v}'\|} \right|$$

which, combined with:

$$|a_i - a_i'| \;=\; \left| \frac{v_i}{\alpha} - \frac{v_i'}{\alpha'} \right| \;=\; \frac{\|\mathbf{v}'\|}{\alpha'}\left| \frac{\alpha'}{\alpha \|\mathbf{v}'\|} v_i - \frac{v_i'}{\|\mathbf{v}'\|} \right| \;=\; \frac{\|\mathbf{v}'\|}{\alpha'}\left| u_i - u_i' + \frac{v_i}{\alpha}\left(\frac{\alpha'}{\|\mathbf{v}'\|} - \frac{\alpha}{\|\mathbf{v}\|} \right) \right|$$

yields:

$$|a_i - a_i'| \;\leqslant\; \frac{\|\mathbf{v}'\|}{\alpha'}\left(|u_i - u_i'| + \|\mathbf{u} - \mathbf{u}'\|_2\, |a_i| \right) \;\leqslant\; \frac{\|\mathbf{v}'\|}{\alpha'}\, (1 + |a_i|)\, \|\mathbf{u} - \mathbf{u}'\|_2$$

from which we finally obtain:

$$|f_\theta(\mathbf{x}) - f_\theta(\mathbf{x}')| \;\leqslant\; \left[K\, \frac{\|\mathbf{v}'\|}{\alpha'} \sqrt{\sum_i (1 + |a_i|)^2} \right] \|\mathbf{u} - \mathbf{u}'\|_2 \;.$$

Note that α' is actually a factor of each coefficient of $\mathbf{v}'$, as the derivative of $f_\theta(\mathbf{x}')$ with respect to any parameter is a chain rule starting with $\frac{\partial f_\theta(\mathbf{x}')}{\partial b} = \sigma'\big|_{\sum_i w_i a_i(\mathbf{x}')+b} = \alpha'$. To bound the term between brackets, the same assumptions as previously are sufficient. One can assume that α and α' are not 0, as, if they are, the problem is of little interest ($\mathbf{u}$ or $\mathbf{u}'$ being then not defined).

G.3 Additional proof detail

The kernel $k_\theta^C(\mathbf{x}, \mathbf{x}')$, by definition, is the L^2 inner product between two unit vectors:

$$k_\theta^C(\mathbf{x}, \mathbf{x}') = \frac{\nabla_\theta f_\theta(\mathbf{x})}{\|\nabla_\theta f_\theta(\mathbf{x})\|} \cdot \frac{\nabla_\theta f_\theta(\mathbf{x}')}{\|\nabla_\theta f_\theta(\mathbf{x}')\|} \;.$$
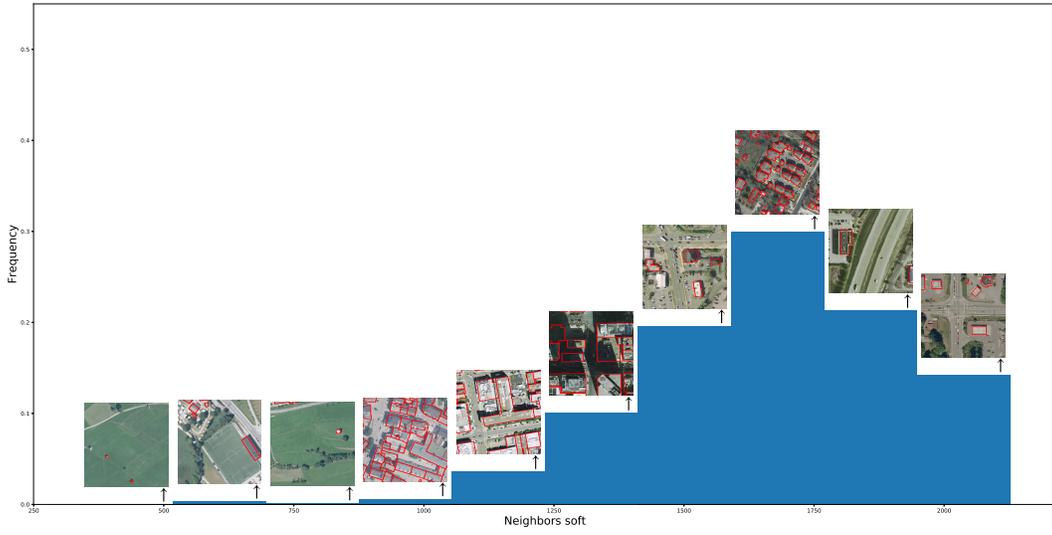
As, for any two unit vectors a and b:

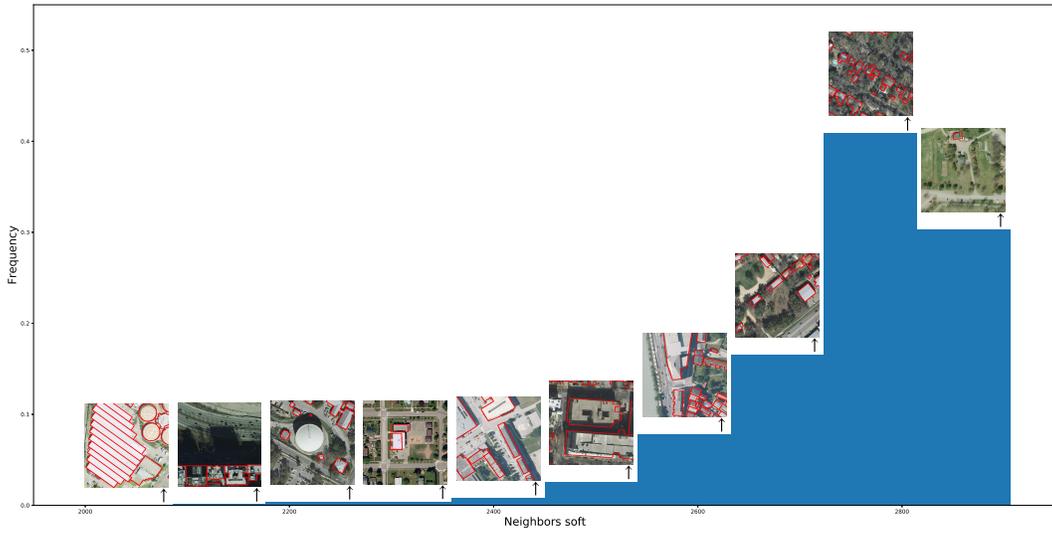$$\|a - b\|^2 \;=\; a^2 + b^2 - 2\, a \cdot b \;=\; 2\, (1 - a \cdot b) \;,$$

we get:

$$\left\| \frac{\nabla_\theta f_\theta(\mathbf{x})}{\|\nabla_\theta f_\theta(\mathbf{x})\|} - \frac{\nabla_\theta f_\theta(\mathbf{x}')}{\|\nabla_\theta f_\theta(\mathbf{x}')\|} \right\| = \sqrt{2}\sqrt{1 - k_\theta^C(\mathbf{x}, \mathbf{x}')} \;.$$
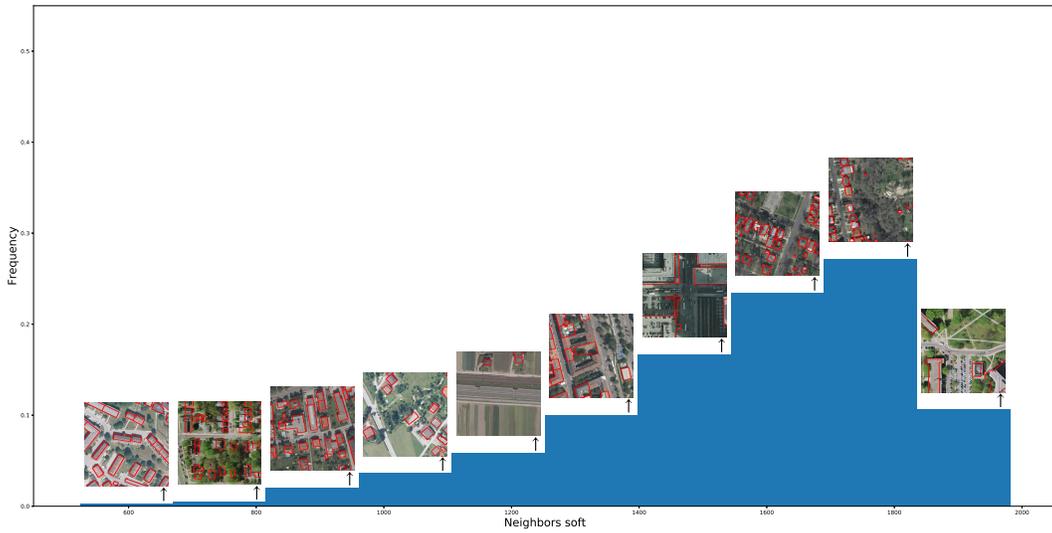
G.4 Data augmentation as a label denoising technique

Data augmentation can be seen as label denoising, as it multiplies the number of neighbors. Indeed, in the infinite sampling limit, where the dataset becomes a probability distribution over all possible images, adding a transformed copy $\mathbf{x}' = T_\phi \mathbf{x}$ of a given point $\mathbf{x}$ (*e.g.* rotating it with an angle α_ϕ and adding small noise ε_ϕ) means adding $(\mathbf{x}', l(\mathbf{x}))$ to the dataset, where $l(\mathbf{x})$ is the desired label for $\mathbf{x}$. But if $(\mathbf{x}', l(\mathbf{x}'))$ was already in the dataset, this amounts to enriching the possible labels for $\mathbf{x}'$. Supposing T_ϕ is an invertible transformation parameterized by ϕ, full data augmentation (*i.e.* for all possible ϕ, applied on all points $\mathbf{x}$) enriches $\mathbf{x}'$ with all labels $l(T_\phi^{-1}(\mathbf{x}'))$. In case of i.i.d. label noise, data augmentation will thus reduce this noise by a factor $\sqrt{\text{number of copies}}$.

(a) Round 1

(b) Round 2

(c) Round 3

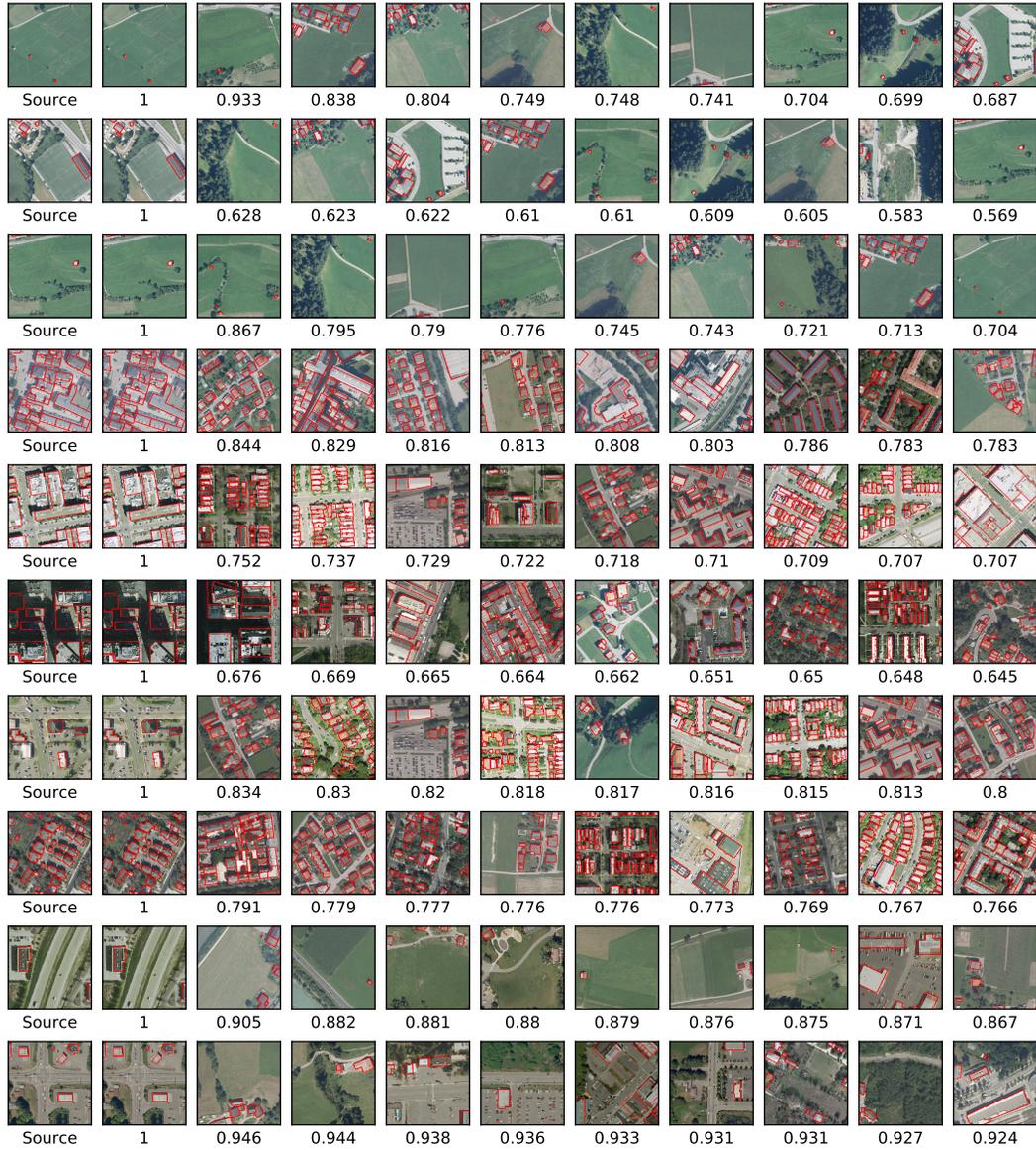Figure 11: Histogram of the soft estimate of neighbors on 3045 patches. Horizontal scale is different for each.

Figure 12: **Round 1**: k-nearest neighbors with k=10. The 10 patches selected correspond to the 10 patches of Fig.11 for that round.
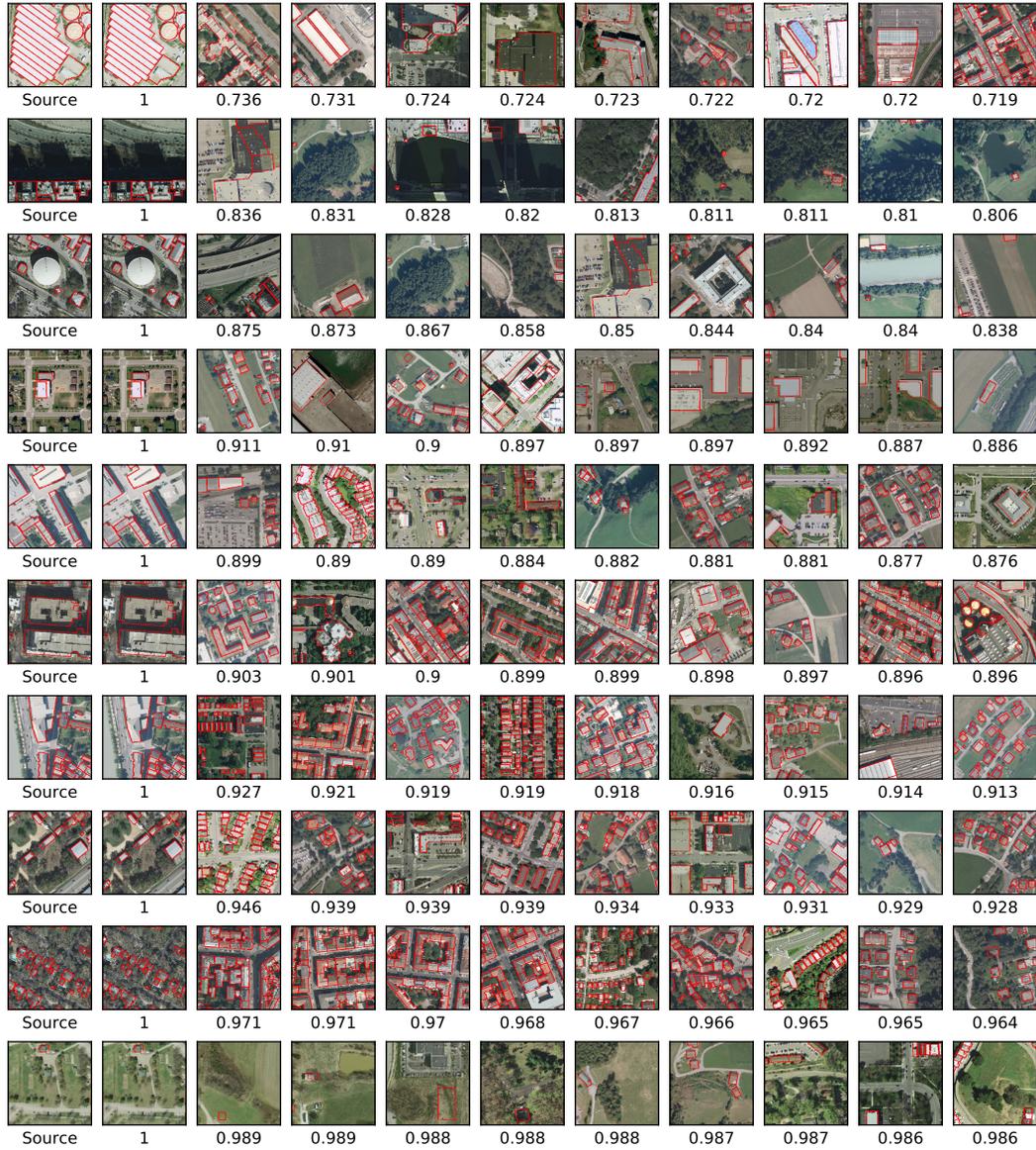
Figure 13: **Round 2**: k-nearest neighbors with k=10. The 10 patches selected correspond to the 10 patches of Fig.11 for that round.
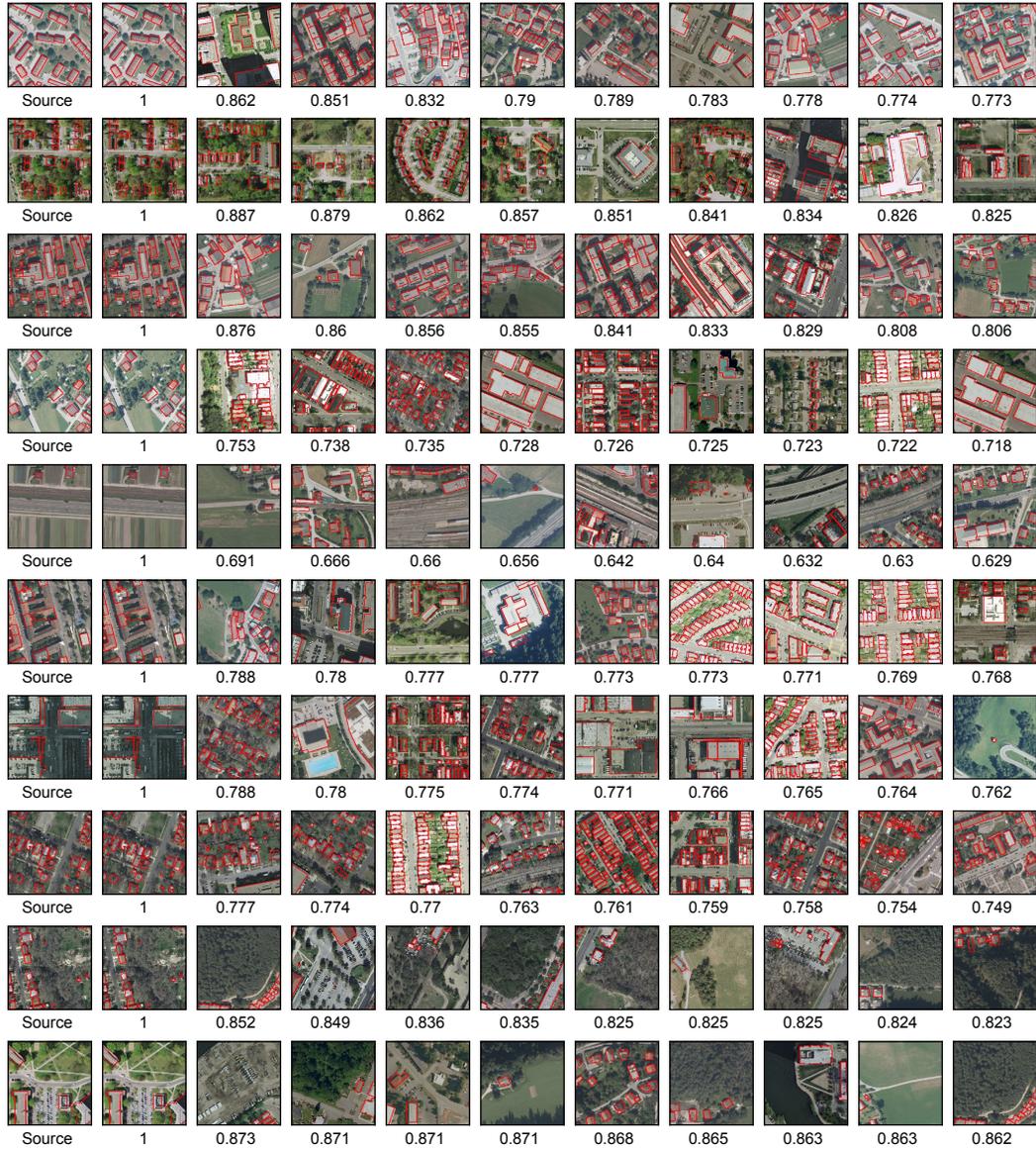
Figure 14: **Round 3**: k-nearest neighbors with k=10. The 10 patches selected correspond to the 10 patches of Fig.11 for that round.
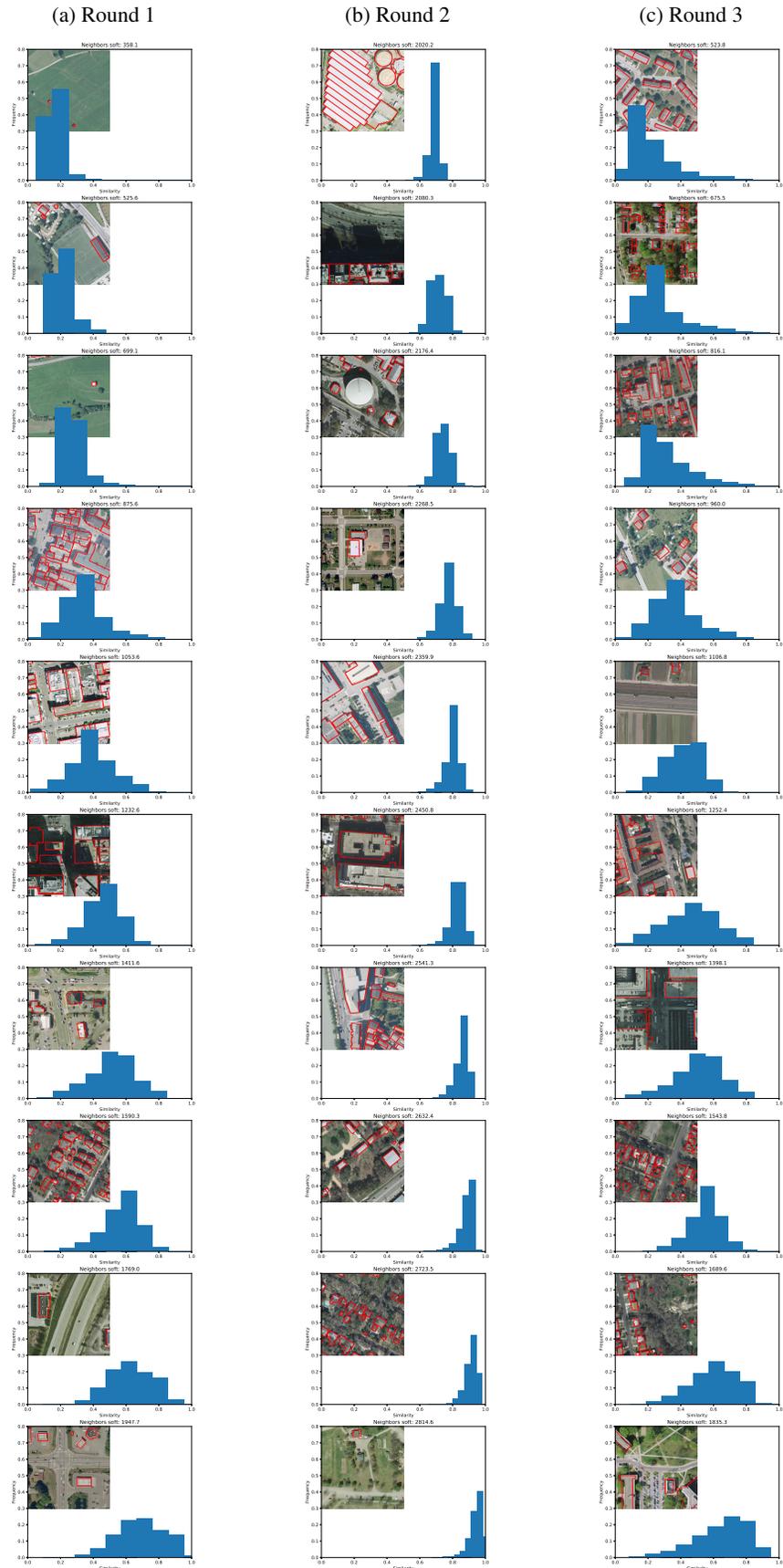
Figure 15: Histograms of similarities shown for the same 10 patches as Fig.11 and Fig.12, 13, 14.

Figure 16: **Round 1**: k-nearest neighbors with k=10. The 10 patches are from from the bloomington22 image. Same patch selection across rounds.

Figure 17: **Round 2**: k-nearest neighbors with k=10. The 10 patches are from from the bloomington22 image. Same patch selection across rounds.
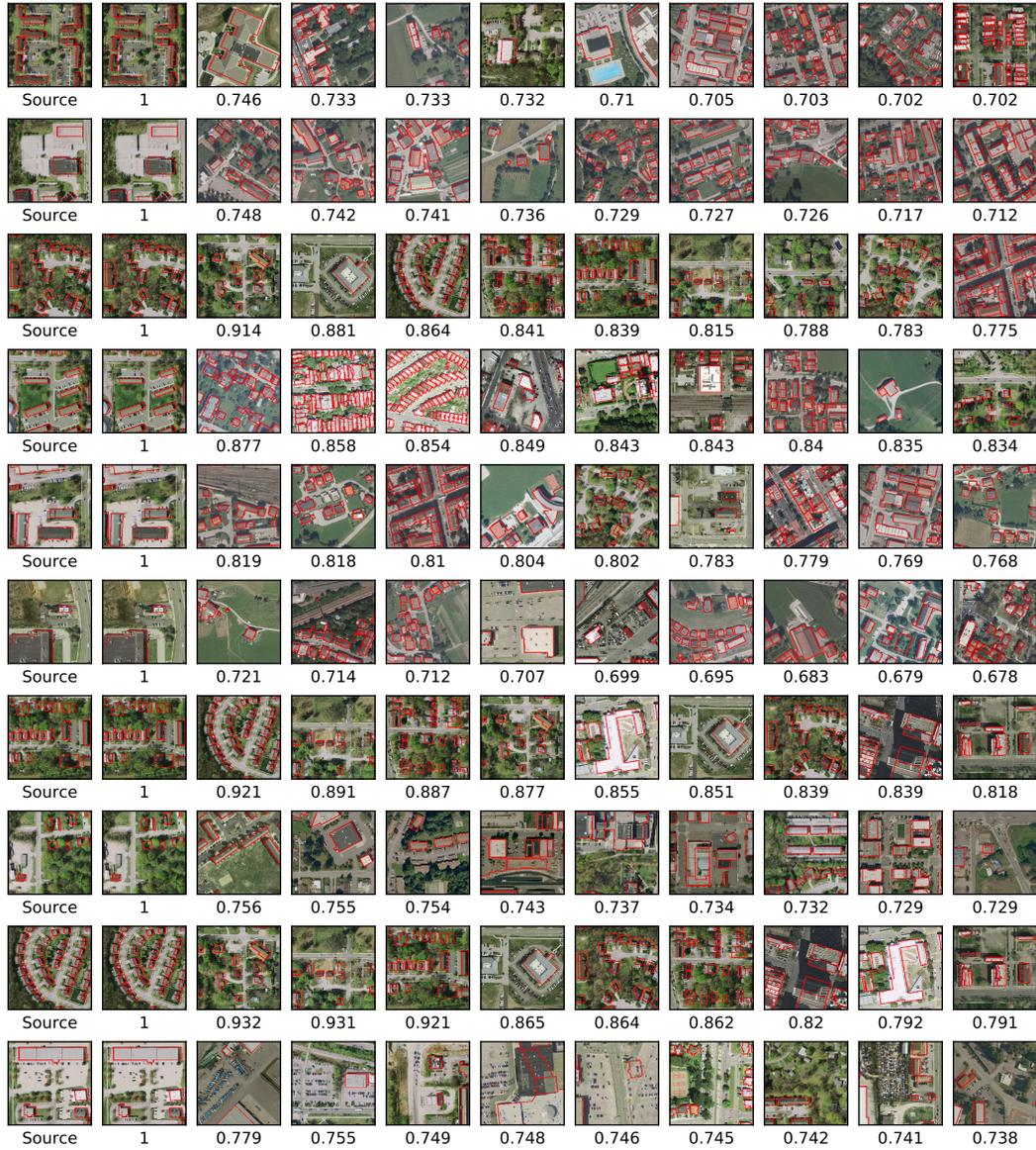
Figure 18: **Round 3**: k-nearest neighbors with k=10. The 10 patches are from from the bloomington22 image. Same patch selection across rounds.
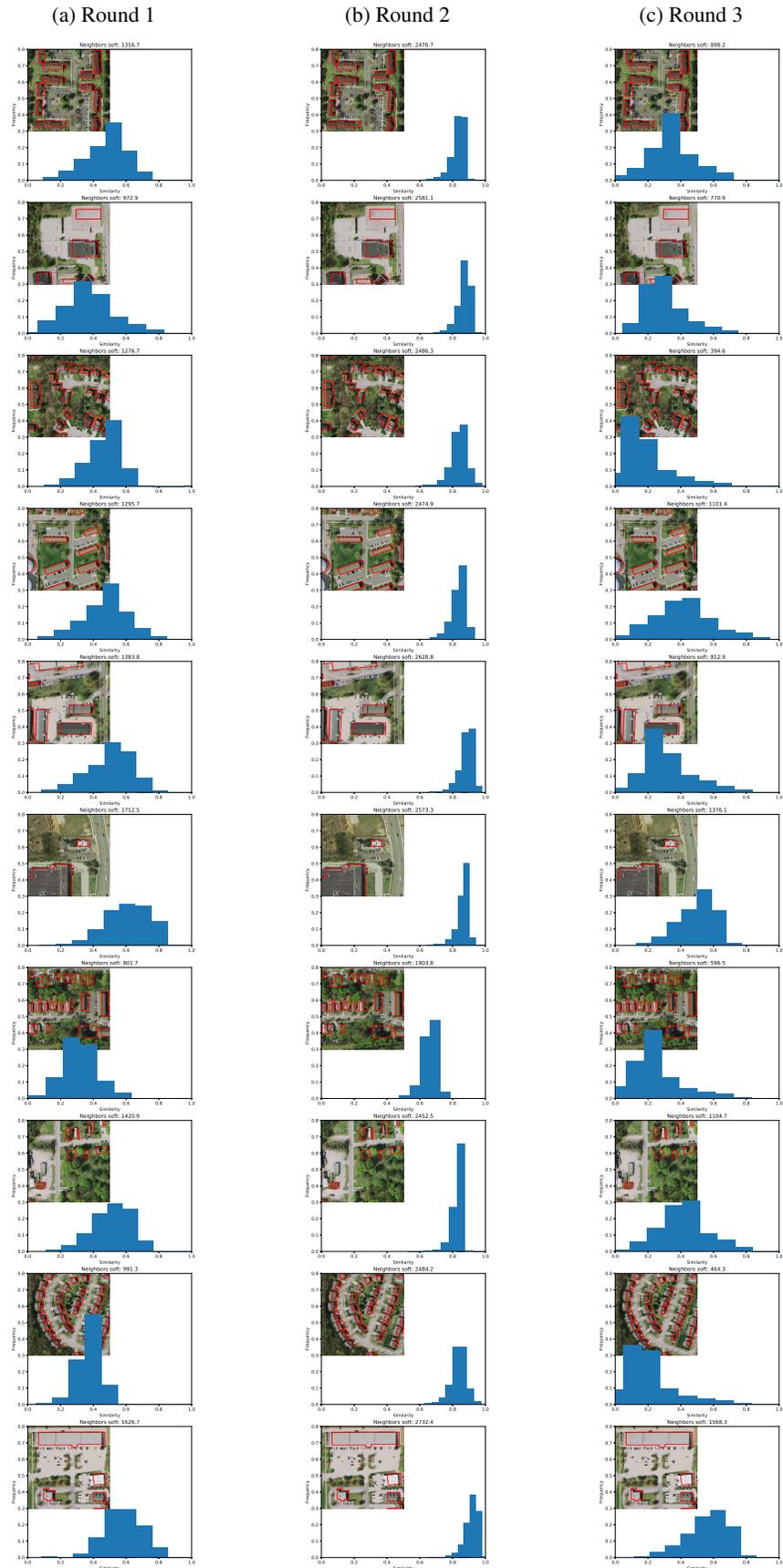
(a) Round 1   (b) Round 2   (c) Round 3

Figure 19: Histograms of similarities shown for the same 10 patches as in Fig.16, 17, 18. Same patch selection across rounds.