Smoothed Differentiation Efficiently Mitigates Shattered Gradients in Explanations

Adrian Hill[1 2] **Neal McKee**[3 4] **Johannes Maeß**[1 2]
Stefan Blücher[1 2] **Klaus-Robert Müller**[1 2 5 6]

[1]BIFOLD – Berlin Institute for the Foundations of Learning and Data, Berlin, Germany
[2]Machine Learning Group, TU Berlin, Berlin, Germany
[3]Bernstein Center for Computational Neuroscience, Berlin, Germany
[4]TU Berlin, Berlin, Germany
[5]Department of Artificial Intelligence, Korea University, Seoul, Korea
[6]Max Planck Institut für Informatik, Saarbrücken, Germany

hill@tu-berlin.de neal@bccn-berlin.de maess@tu-berlin.de
bluecher@tu-berlin.de klaus-robert.mueller@tu-berlin.de

Abstract

Explaining complex machine learning models is a fundamental challenge when developing safe and trustworthy deep learning applications. To date, a broad selection of explainable AI (XAI) algorithms exist. One popular choice is SmoothGrad, which has been conceived to alleviate the well-known shattered gradient problem by smoothing gradients through convolution. SmoothGrad proposes to solve this high-dimensional convolution integral by sampling – typically approximating the convolution with limited precision. Higher numbers of samples would amount to higher precision in approximating the convolution but also to higher computing demand, therefore in practice only few samples are used in SmoothGrad. In this work we propose a well founded novel method *SmoothDiff* to resolve this tradeoff yielding a *speedup of over two orders of magnitude*. Specifically, *SmoothDiff* leverages automatic differentiation to decompose the expected values of Jacobians across a network architecture, directly targeting only the non-linearities responsible for shattered gradients and making it easy to implement. We demonstrate SmoothDiff's excellent speed and performance in a number of experiments and benchmarks. Thus, SmoothDiff greatly enhances the usability (quality and speed) of SmoothGrad – a popular workhorse of XAI.

1 Introduction

XAI has in the past years become a popular resource to further a user's understanding of various machine learning models such as neural networks, kernel methods, LSTMs and transformers, see e.g. [1]. Among others, XAI allows to debug models, scrutinize data sets and find Clever-Hans effects in supervised and unsupervised models [2, 3]. Post-hoc explanation methods, see [1], can e.g. be based on gradients [4, 5, 6], first- and higher-order Taylor expansions around meaningful root points [7, 8, 9, 10], and Shapley values [11, 12].

In this work we will place our focus on the inner workings of a classifier by inspecting the gradient of its prediction with respect to its input features. Usually, the focus is on the gradient $\nabla f_c(\boldsymbol{x})$ of the maximally activated output logit, however any other logit can be inspected as well, resulting in an explanation for that respective class (see Appendix A for notation). We limit ourselves to the fixed-model regime, in which the model cannot be retrained for the purpose of interpretability.

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

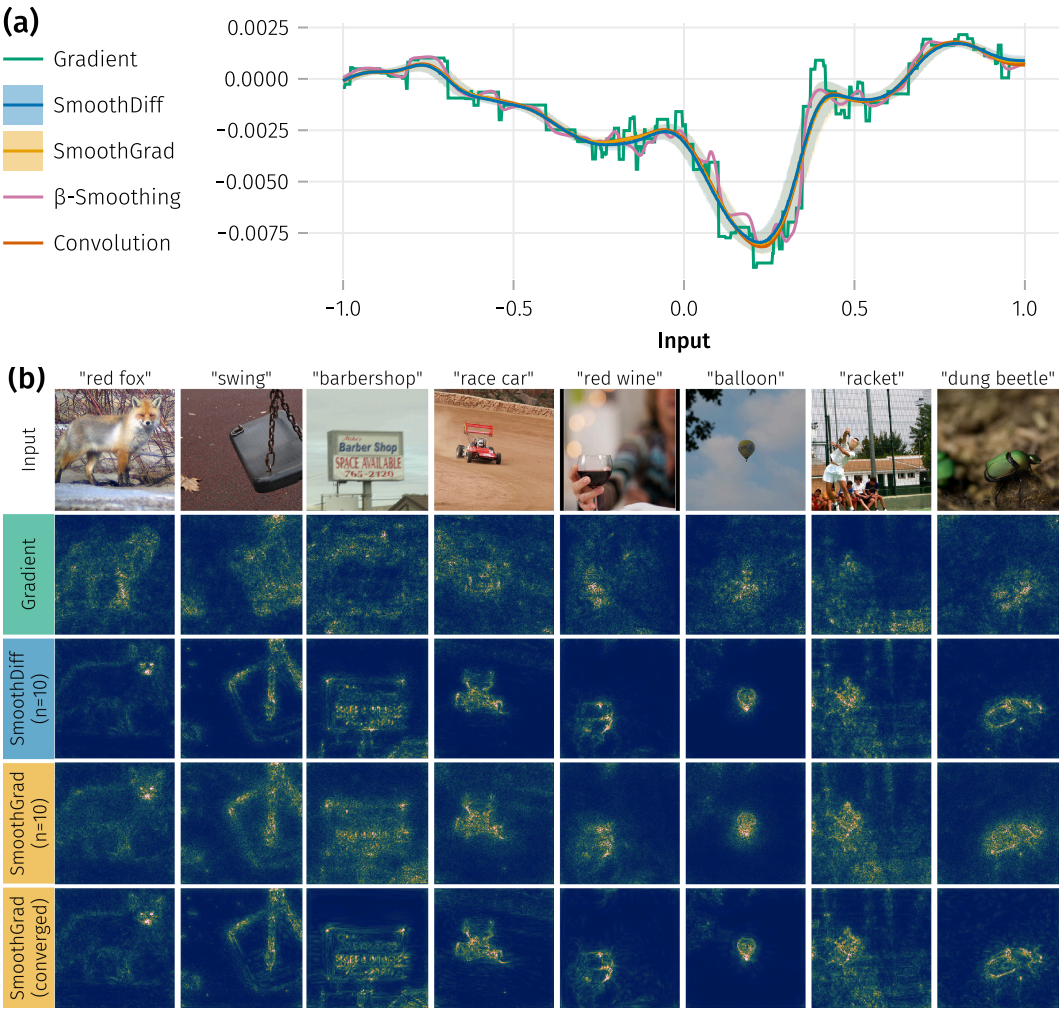

Figure 1: **(a)** Visualization of the shattered gradient problem on a 1D deep ReLU network. Gaussian convolution removes high-frequency components of the shattered gradient and is well approximated by both SmoothDiff and SmoothGrad. **(b)** On high-dimensional ImageNet models, SmoothGrad suffers from low sample efficiency, whereas at just 10 samples, SmoothDiff's level of noise reduction matches a fully converged SmoothGrad. See Appendix B for implementation details.

1.1 Preliminaries

Shattered gradient problem. With growing depth of neural networks, the vector field of gradients over the input space $\mathcal{X}$ gets increasingly noisy, as shown in [13]. For small changes in inputs $\boldsymbol{x} \in \mathcal{X}$, resulting gradient vectors $\nabla f_c(\boldsymbol{x})$ vary strongly, resembling white noise. This phenomenon has been termed the *shattered gradient problem* (SGP), namely the SGP renders them uninformative due to a bad signal to noise ratio, as shown in Figure 1: For a 1-dimensional toy example we observe that the SGP results in large step-wise fluctuations of the gradient; also for gradient explanations in vision models (here VGG-19 [14]), the SGP leads to highly noisy heatmaps.

Gaussian convolutions. Since the SGP closely resembles white noise in deep networks, it motivates the use of a Gaussian filter in the input domain to suppress high-frequency signals: if images are perceptually robust to small amounts of noise, explanations should also be stable with respect to minor input variations. This implies that explanations should not vary rapidly and have little high-frequency content in the input domain. Mathematically, the desired Gaussian filter operation corresponds to a convolution with a Gaussian kernel g over the input domain $\mathcal{X}$. This in turn is equal to the expected value

$$(\nabla f_c * g)(\boldsymbol{x}) = \int_{\mathcal{X}} \nabla f_c(\hat{\boldsymbol{x}}) g(\boldsymbol{x} - \hat{\boldsymbol{x}}) d\hat{\boldsymbol{x}} = \mathbb{E}_{\hat{\boldsymbol{x}} \sim \mathcal{N}_g}[\nabla f_c(\hat{\boldsymbol{x}})] \,, \tag{1}$$

where $\mathcal{N}_g$ is the Normal distribution analogous to the kernel g in terms of covariances [15]. The variance of this distribution represents a trade-off between the strength of the smoothing and using information from regions of the input space that are no longer relevant to the actual input. It should be chosen large enough to aggregate gradient information over similar inputs, but small enough to not change the classification or subjective qualitative nature of the input to humans.

SmoothGrad. While the convolution in (1) is generally intractable or at least unfeasible, it can be approximated via sampling [6]. Using n samples $\hat{\boldsymbol{x}}_i$ drawn from $\mathcal{N}_g$, we obtain the definition of SmoothGrad (SG) as

$$\mathrm{SG}_n(\boldsymbol{x}) = \frac{1}{n} \sum_{i=1}^{n} \nabla f_c(\hat{\boldsymbol{x}}_i) \,. \tag{2}$$

Since this corresponds to an averaging of input gradients over noisy inputs, the computational cost of SG grows linearly in the amount of samples, requiring n forward and n backward passes through the model, with a suggested sample size of $n = 50$ [6]. While SG converges to the convolution (1) in the infinite sample limit, it is well-known that computationally tractable amounts of samples may not be sufficient in practice, say, for ImageNet samples, as is visually apparent in the heatmaps of Figure 1 (b) and demonstrated in Section 3.1. Due to this, SmoothGrad is commonly seen as a *perturbation-based* method instead of one based on Gaussian convolution.

Gradient explanations rewritten as factorized Jacobians. For any function $f : \mathbb{R}^n \to \mathbb{R}^m$, the gradient of the c-th output corresponds to the c-th row of the Jacobian $\boldsymbol{J}_f(\boldsymbol{x})$. For a deep neural network $f = f^N \circ f^{N-1} \circ ... \circ f^1$ composed of N functions f^n, this gradient can therefore be derived via the chain rule as

$$\begin{aligned} \nabla f_c(\boldsymbol{x})^T = \left[\boldsymbol{J}_f(\boldsymbol{x})\right]_{c,:} &= \boldsymbol{e}_c^T \cdot \boldsymbol{J}_f(\boldsymbol{x}) \\ &= \boldsymbol{e}_c^T \cdot \boldsymbol{J}_{f^N}(\boldsymbol{a}^N) \cdot \boldsymbol{J}_{f^{N-1}}(\boldsymbol{a}^{N-1}) \cdot ... \cdot \boldsymbol{J}_{f^1}(\boldsymbol{x}) \,, \end{aligned} \tag{3}$$

where we denote the input activation of the n-th function as $\boldsymbol{a}^n$, such that the input $\boldsymbol{x} = \boldsymbol{a}^1$ and $\boldsymbol{a}^{n+1} = f^n(\boldsymbol{a}^n)$. We would like to stress that such gradients can efficiently be computed using reverse mode automatic differentiation (AD), sequentially accumulating the *vector-Jacobian products* (VJPs) on the right-hand side of (3) from the left to the right [16, 17, 18]. Since this computation requires knowledge of all activations $\boldsymbol{a}^N$ to $\boldsymbol{a}^1$, a forward pass through the model must first be computed, saving all activations in memory for the subsequent backward pass. By writing the forward pass in AD frameworks such as PyTorch [19] and JAX [20], gradients explanations (3) can be computed up to numerical accuracy without requiring any additional code.

1.2 Contributions

In this work, we provide the following key contributions:

1. We motivate Gaussian convolutions in embedding space as a solution to the shattered gradient problem [13], emphasize that SmoothGrad numerically approximates said convolution in the infinite sample limit, and demonstrate that commonly used SmoothGrad sample sizes are typically insufficient.
2. We introduce SmoothDiff based on AD, which is both faster and more sample efficient than SmoothGrad in approximating Gaussian convolutions. We provide experimental benchmarks and clear guidelines on how to implement SmoothDiff for arbitrary differentiable model architectures.
3. We demonstrate that SmoothDiff directly and efficiently addresses the shattered gradient problem.

1.3 Related work

NoiseGrad is a variant of SG that computes an average of gradient explanations over perturbed model weights instead of perturbed inputs [21]. It is therefore not connected to input space convolutions.

Perturbing both inputs and model weights results the *FusionGrad* method. In [22], a low-pass filter is applied to gradient explanations in image space. Since the smoothing is not applied in input space, this approach does not directly address the SGP and runs into the risk of removing meaningful high-frequency spatial information from heatmaps. In [23], the high curvature of neural network decision boundaries and how it can be exploited to manipulate explanations is explored. The proposed β-*Smoothing* method smoothes the decision boundary by replacing ReLUs with Softplus activations on the backward pass. As we show in Appendix G, this is a good approximation of a Gaussian convolution in the case of a single-layer neural network. However, as shown in Figure 1(a), the approximation error grows with network depth, as after the first layer, subsequent activations may not be Gaussian anymore.

2 SmoothDiff

We now propose SmoothDiff, a computationally efficient approximation of the Gaussian convolution. Inserting the factorized neural network gradient (3) into the convolution (1), we obtain

$$(\nabla f_c * g)(\boldsymbol{x}) = \boldsymbol{e}_c^T \cdot \mathbb{E}_{\hat{\boldsymbol{x}}\sim\mathcal{N}_g}\left[\boldsymbol{J}_{f^N}(\hat{\boldsymbol{a}}^N)\cdot\boldsymbol{J}_{f^{N-1}}(\hat{\boldsymbol{a}}^{N-1})\cdot\ldots\cdot\boldsymbol{J}_{f^1}(\hat{\boldsymbol{x}})\right], \tag{4}$$

in which all intermediate activations $\hat{\boldsymbol{a}}^i$ deterministically depend on the random variable $\hat{\boldsymbol{x}}$. By neglecting cross-covariances between Jacobians, this expected value of a product of Jacobians can be approximated as a product of expected values of Jacobians, resulting in the definition of SmoothDiff:

$$\mathrm{SD}(\boldsymbol{x}) = \boldsymbol{e}_c^T \cdot \underbrace{\mathbb{E}_{\hat{\boldsymbol{a}}^N}\left[\boldsymbol{J}_{f^N}(\hat{\boldsymbol{a}}^N)\right]}_{=:\widehat{\boldsymbol{J}_{f^N}}} \cdot \ldots \cdot \underbrace{\mathbb{E}_{\hat{\boldsymbol{x}}}\left[\boldsymbol{J}_{f^1}(\hat{\boldsymbol{x}})\right]}_{=:\widehat{\boldsymbol{J}_{f^1}}}. \tag{5}$$

Note that this definition is based on the gradient decomposition (3), however, here we propose to *replace* Jacobians $\boldsymbol{J}_{f^i}$ by the expected values of Jacobians $\widehat{\boldsymbol{J}_{f^i}}$.

2.1 Jacobian factorization

The chain rule can be used to factorize neural network Jacobians to arbitrary levels of granularity: from layer-wise Jacobians down to derivatives of elementary functions like addition and multiplication. For SmoothDiff, this raises the question of optimal Jacobian factorization (5), as this choice affects which cross-covariance terms are neglected, trading an approximation bias for increased sample efficiency and computational performance. While in theory many options arise, in practice the choice of factorization for a given model architecture is typically straightforward.

We provide the following intuition: Since neural networks $f : \mathbb{R}^n \to \mathbb{R}^m$ are universal function approximators, their Jacobians–which correspond to linearizations of said universal function–can be arbitrary $m \times n$ matrices. Instead of estimating such "unconstrained" network Jacobians directly (2), sample efficiency can be improved by decomposing the network down to functions with constrained Jacobian structure. This includes:

1. **Linear and affine functions**. Since their Jacobians are constant, $\hat{\boldsymbol{J}} = \boldsymbol{J}$ is known.
2. **Functions with sparse Jacobians** (i.e. containing mostly zeros). Examples include vectorized scalar functions, such as element-wise activation functions, whose Jacobians are diagonal matrices: Since the i-th output only depends on the i-th input, $\hat{J}_{i,j} = J_{i,j} = \frac{\partial f_i}{\partial x_j} = 0$ for $i \neq j$.
3. **Functions with bounded partial derivatives** $\frac{\partial f_i}{\partial x_j}$. This property appears in common activation functions. For ReLU, softplus and $\tanh$, it holds that $0 \leq \frac{\partial f}{\partial x} \leq 1$.
4. **Functions with a closed-form analytical solution to the expected Jacobian** $\hat{\boldsymbol{J}}$. Point 1 is a special case of this.

Applied to common network architectures, these principles discourage us from naively applying the factorization on a layer-wise basis. Instead, element-wise activation functions should be separated from otherwise affine functions. The Jacobian of a dense layer $f(\boldsymbol{x}) = \sigma(g(\boldsymbol{x})) = \sigma(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$ should therefore be decomposed as

$$\boldsymbol{J}_f = \boldsymbol{J}_\sigma \cdot \boldsymbol{J}_g = \boldsymbol{J}_\sigma \cdot \boldsymbol{W}. \tag{6}$$

In other words: instead of estimating the dense matrix $\widehat{\boldsymbol{J}}_f$ containing $m \cdot n$ entries, where n is the input and m the output dimensionality, we should leverage our knowledge of the constant Jacobian $\widehat{\boldsymbol{J}}_g = \boldsymbol{J}_g = \boldsymbol{W}$ and only estimate the m bounded entries in the sparse diagonal matrix $\widehat{\boldsymbol{J}}_\sigma$, resulting in a higher sample efficiency. We give an implementation example of this in the following.

2.2 Example implementation

SmoothDiff can be implemented as a *modified-backprop method*,[1] replacing the sequential computation of VJPs (3) by what we term vector-*expected-Jacobian* products (VEJPs) in (5), sequentially evaluating $\boldsymbol{v}^T \widehat{\boldsymbol{J}}_{f^n}$ on a single backward pass. For linear and affine functions with expected Jacobian $\hat{\boldsymbol{J}} = \boldsymbol{J}$, it follows that $\boldsymbol{v}^T \hat{\boldsymbol{J}} = \boldsymbol{v}^T \boldsymbol{J}$. Applying SmoothDiff to such functions requires no additional code, as common AD frameworks such as PyTorch and JAX already implement corresponding VJPs. SmoothDiff only requires implementing VEJPs for functions in the model which are non-linear with respect to their input, such that $\hat{\boldsymbol{J}} \neq \boldsymbol{J}$. If the model contains functions for which $\hat{\boldsymbol{J}}$ has no analytical solution, sampling estimates can be obtained by computing n forward passes through the model. As a motivating example, we now discuss how to implement a VEJP for ReLUs σ. This should provide a blueprint on how to implement SmoothDiff on a variety of current and future model architectures.

As discussed in the previous section, the Jacobian $\boldsymbol{J}_\sigma$ of any element-wise activation function is a diagonal matrix. For ReLUs specifically, the diagonal entries correspond to

$$J_{\sigma_{i,i}}(\boldsymbol{a}) = \mathbb{1}_{x \geq 0}(a_i) := \begin{cases} 1 \text{ if } a_i \geq 0 \\ 0 \text{ else} \end{cases}, \tag{7}$$

where $\mathbb{1}_{x \geq 0}$ is the indicator function returning one for inputs greater or equal zero. From this, we can derive the expected Jacobian $\widehat{\boldsymbol{J}}_\sigma$ as

$$\widehat{J}_{\sigma_{i,i}} = \mathop{\mathbb{E}}_{\hat{\boldsymbol{a}}}\left[J_{\sigma_{i,i}}(\hat{\boldsymbol{a}})\right] = \lim_{n \to \infty} \underbrace{\frac{1}{n} \sum_{j=1}^{n} \mathbb{1}_{x \geq 0}(a_i)}_{=: p_i}, \tag{8}$$

where the right-hand side corresponds to the expected ratio of positive activations. A sampling approximation of this term can be obtained by computing a finite amount of n forward passes through the model with inputs $\hat{\boldsymbol{x}}$ drawn from $\mathcal{N}_g$, while keeping track of the count of non-zero activations p_i. For a given vector $\boldsymbol{v}$, this results in the VEJP

$$\left[\boldsymbol{v}^T \widehat{\boldsymbol{J}}_\sigma\right]_i = v_i \widehat{J}_{\sigma_{i,i}} = v_i \frac{p_i}{n}, \tag{9}$$

such that the SmoothDiff explanation (5) can be computed in a single matrix-free backward pass. An analogous approach can be taken to estimate $\hat{\boldsymbol{J}}$ of max pooling layers, whose entries correspond to binary indicator functions of maximally activated inputs, requiring a sampling estimate of the ratio of activations with maximum value. Pseudocode implementation for both ReLUs and max pooling layers are given in Appendix I. While our exposition emphasized sequential models for the sake of simplicity, more complex, branching models such as ResNets [24] are automatically handled by AD systems. In fact, implementing the two aforementioned VEJPs is all it takes to run SmoothDiff on common vision models such as VGG and ResNets.

2.3 Fallback VEJP

SmoothDiff can be applied to arbitrary differentiable models. For any function (e.g. an arbitrary layer), a fallback VEJP can be implemented that samples n forward passes, computes the full Jacobian for each sample, and approximates the expected value by keeping track of the running mean. The naive computation of a Jacobian requires either as many VJPs as the function has outputs (reverse mode) or as many *Jacobian-vector products* (JVPs) as it has inputs (forward mode). If the Jacobian exhibits sparsity, *automatic sparse differentiation* [17, 25] can drastically reduce the number of required VJPs/JVPs (see [26, 27] for applications in deep learning).

[1] The term *modified-backprop method* is a misnomer based on implementation details. Backpropagation is a special case of reverse mode AD, while Jacobian transformations (such as our expected Jacobians) can in many cases also be implemented for forward mode AD systems. Refer to [17] and [15] for an overview of AD techniques.

For functions with diagonal Jacobians (e.g. activation functions) this fallback VEJP is efficient, only requiring a single VJP per sample. For large and dense Jacobians however, it requires a lot of compute and memory, negating SmoothDiff's performance benefits over SG.

2.4 Computational performance and other properties

As we will demonstrate in Section 3.1, estimating factorized Jacobians according to Section 2.1 is significantly more sample efficient than estimating the Jacobian of the entire neural network. In practice, SmoothDiff therefore requires an order of magnitude fewer samples than SG to achieve a visually comparable (see Section 3.3 and Appendix K) amount of smoothing or noise reduction.

Additionally, for single class explanations of sample size n, SmoothDiff only requires n forward and a single backward pass, whereas SG requires n forward and n backward passes. Denoting the unit time complexity of a single forward pass $f(\boldsymbol{x})$ by τ, the corresponding backward pass has been shown to be in the same order of time complexity $O(\tau)$ [28], and we (conservatively) estimate their costs to be equal. For single-class explanations with n samples, SmoothDiff yields a stark improvement requiring $n + 1$ (forward or backward) passes in comparison to SG's $2n$.

The largest performance gains occur when multiple explanations need to be computed for a given input, e.g. multi-class explanations and explanations with respect to concepts or filters that restrict the propagation through specific neurons. This is required for spectral analysis of explanations [29], concept relevance propagation [30], and higher-order explanation techniques [9, 10, 31]. So far, such methods have predominantly used the highly efficient LRP method [7, 8]. Assuming explanations of k classes (or concepts/filters), LRP requires only a single forward pass and k backward passes. In contrast, SG only estimates class-dependent smoothed gradients (2) and needs to compute n forward and n backward passes for each of the k classes, resulting in a total of $2nk$ passes.
Since SmoothDiff estimates the full smoothed Jacobian (5) using matrix-free operators (VEJPs), it can be applied at a similar time complexity to LRP. Once VEJPs are estimated in n forward passes, they don't need to be recomputed and only one backward pass per class (or concept/filter) is needed, resulting in a total of $n + k$ passes. Assuming an explanation of all $k = 1000$ ImageNet classes using $n = 50$ samples,[2] SmoothDiff yields a computational speedup of two orders of magnitude[3] over SG's approximation to Gaussian filtering and approaches three orders of magnitude including the increased sample efficiency demonstrated in Section 3.

In addition to convergence and performance improvements, SmoothDiff directly addresses the SGP, as we demonstrate in Appendix H. We further discuss in Appendix J how SmoothDiff's expected network Jacobian increases the *rank* compared to the regular network Jacobian $\boldsymbol{J}_f$, a desirable property for explanation methods that has been found to allow for more expressiveness and higher variation of explanations across classes [29].

2.5 Limitations

While SmoothDiff can be applied to arbitrary models, performance benefits are obtained by deriving performant VEJP (see Section 2.3). Such VEJPs have yet to be derived for attention layers, currently negating SmoothDiff's performance benefits over SG on ViT [32]. This limitation could be addressed through partial application of SmoothDiff, falling back to VJPs when no VEJPs are available.

3 Experiments

We evaluate SmoothDiff and SG on a pre-trained VGG-19 model [14] and the ImageNet dataset [33]. All experiments were run on a *NVIDIA A100 80GB* GPU and *AMD EPYC 9124* CPU. We use the *Julia-XAI* ecosystem [34, 35], *Flux.jl* deep learning framework [36, 37] and implement VEJPs using *ChainRules.jl* [38] and *Zygote.jl* [39].

We provide the complete source code to reproduce our experiments at `https://github.com/adrhill/smoothdiff-experiments`, including SmoothDiff reference implementations in Julia [40] and PyTorch.

[2] The suggested sample size for SG [6].

[3] SmoothDiff requires $n + k = 1050$ forward and backward passes, whereas SG requires $2nk = 10^5$.

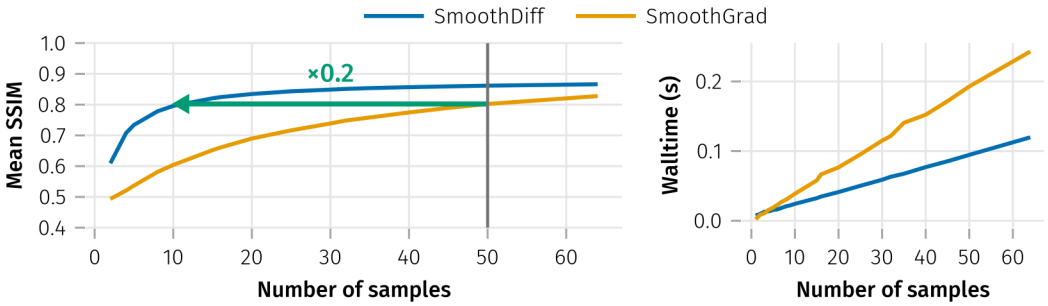

Figure 2: SmoothDiff improves on SG in terms of both *sample efficiency* and *computational performance*. **Left**: Improvements in sample efficiency, measured by the similarity to converged SG explanations in terms of SSIM [41]. At the recommended sample size of 50, SG achieves a SSIM of 0.8. SmoothDiff requires five times fewer samples to achieve a similar SSIM, highlighted by the green arrow. **Right:** Improvements in computational efficiency, measured by walltime per sample. Both SmoothDiff and SG scale linearly, however SmoothDiff is approximately twice as fast across sample sizes.

3.1 Convergence comparison

Since the Gaussian convolution (1) has no analytical solution, we use converged[4] SG explanations over a batch of 128 randomly drawn ImageNet [33] images as our reference. Convergence towards this reference is quantified using the *structural similarity index measure* (SSIM) [41]. A standard deviation of $\sigma = 0.5$ is used for the sampling distribution $\mathcal{N}_g$ in both SmoothDiff and SG.

The resulting mean SSIM are shown on the left of Figure 2. Since SG is used as a reference, it by construction approaches a SSIM of 1 with increasing sample counts. We observe that through the factorization in (5), SmoothDiff trades a small inherent bias in form of neglected cross-covariance terms for a large increase in sample efficiency. Using the recommended sample size of $n = 50$, SG reaches a SSIM of 0.8, whereas with SmoothDiff, the same SSIM is obtained in just 10 samples, resulting in increased sample efficiency of factor 5. For a discussion of the neglected cross-covariance terms, refer to Appendix E.

3.2 Benchmarks

We benchmark the performance of SmoothDiff and SG as a function of sample size n by computing explanations for batches of 128 ImageNet images. As shown on the right side of Figure 2, SmoothDiff is approximately twice as fast as SG across all sample sizes, further compounding the increased sample efficiency demonstrated in Section 3.1. These computational performance improvements match theoretical considerations in Section 2.4, where single-class explanations using SmoothDiff are shown to require $n + 1$ forward and backward passes, whereas SG requires $2n$.

3.3 Heatmaps

For a qualitative comparison of SmoothDiff and SG convergence behavior we visualize heatmaps over increasing sample sizes in Figure 3. Even at small sample sizes, SmoothDiff leads to well-contoured heatmaps with little noise, while SG only progressively reduces noise effects as a function of n; it becomes apparent from Figure 3 that even at the recommended parametrization of SG, i.e. $n = 50$, there is further room for improvement. Further heatmaps including examples of multi-class explanations can be found Appendix K.

3.4 Quantitative results

In addition to the qualitative results in Section 3.3, we use pixel-flipping to quantify SmoothDiff's performance. In the absence of a ground truth, pixel-flipping measures the *faithfulness* of an XAI method by perturbing features in the order of their attribution strength. We use the *symmetric*

[4]More specifically, SmoothGrad explanations using $n = 10^6$ samples.

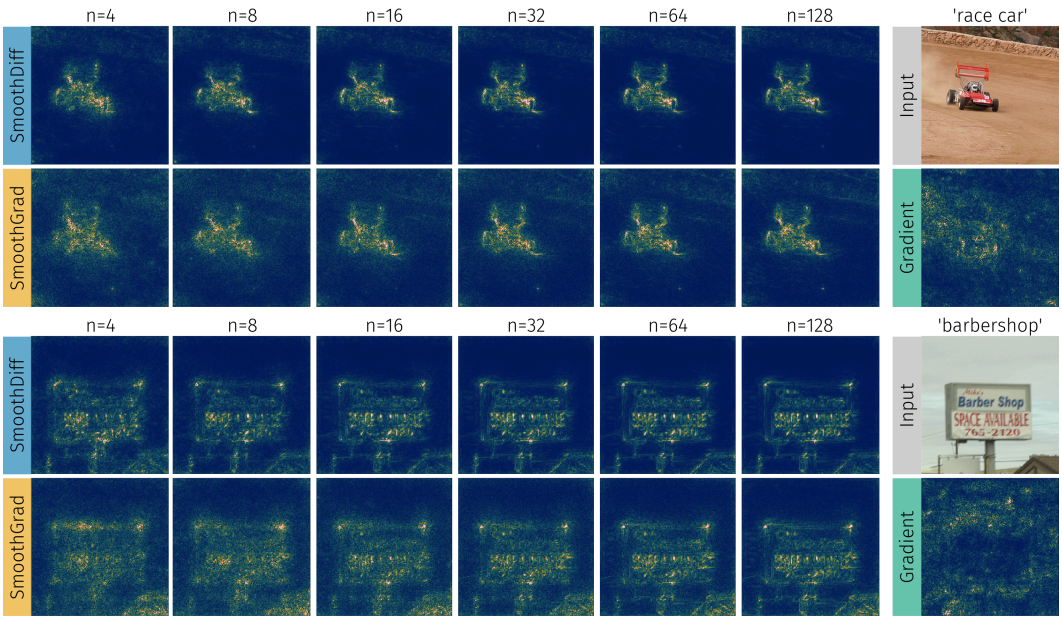

Figure 3: Qualitative comparison of SmoothDiff and SG heatmap convergence over increasing sample sizes. Even at very small sample sizes, SmoothDiff leads to well-contoured heatmaps with little noise.

relevance gain (SRG) measure [42], which largely removes the influence of imputing methods on pixel-flipping. This is achieved by computing the area under two pixel-flipping curves: one deleting features in order of *most influential first* (MIF), the other in order of *least influential first* (LIF).[5] The SRG is defined as the difference between the LIF and MIF measures. Figure 4 shows the results using mean imputing and a standard deviation of $\sigma = 0.5$ for SmoothDiff and SG. SmoothDiff reaches a higher peak SRG and does so at a lower sample size. It remains superior or equal to SmoothGrad up to approximately the recommended sample size of SmoothGrad.

We additionally compare optimal SmoothDiff and SmoothGrad settings from Figure 4 to regular gradient explanations and β-Smoothing (see Section 1.3 and Appendix G), which also attempts to smooth gradients. The pixel-flipping results are summarized in Table 1, demonstrating that SmoothDiff outperforms other methods in the SRG, as well as the LIF and MIF measures [42].

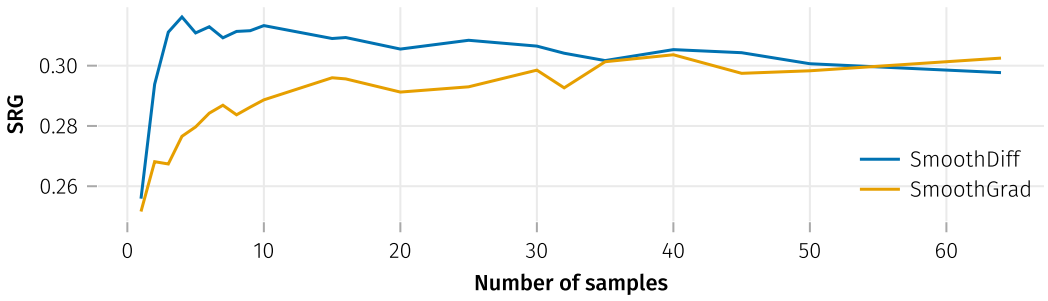

Figure 4: Comparison of SmoothDiff and SG in terms of pixel-flipping performance, measured by SRG [42]. While the performance of SG increases with sample size up to $n = 40$, SmoothDiff's performance peaks at just $n = 4$ samples.

[5] In terms of the area under the pixel-flipping curve, deleting LIF is equivalent to inserting MIF. The SRG measure therefore combines insertion and deletion tests.

Table 1: Pixel-flipping results (↑: higher is better, ↓: lower is better).

Method	Parameters	LIF ↑	MIF ↓	SRG ↑
SmoothDiff	$\sigma = 0.5, n = 4$	**0.382**	**0.066**	**0.316**
SmoothGrad	$\sigma = 0.5, n = 40$	0.373	0.069	0.304
β-Smoothing	$\beta = 1.0$	0.316	0.086	0.230
β-Smoothing	$\beta = 2.0$	0.311	0.093	0.218
β-Smoothing	$\beta = 0.5$	0.293	0.098	0.195
Gradient	—	0.276	0.111	0.166

For additional quantitative results in terms of *localization*, *complexity* and *robustness*, refer to Appendix F, which compares SmoothDiff with gradients, SG, *SmootGrad-Squared* [43], *Integrated Gradients* [44], *LRP composites* [45, 46], *GradCAM* [47], and several random baselines.

4 Discussion

SmoothGrad is a popular gradient-based choice for explaining ML models based on convolving gradients. While originally conceived for compensating shattered gradient effects, it is challenged by the need to accurately approximate the convolution, which would, in principle, require expensive sampling. In practice, SmoothGrad users compromise by trading accuracy with speed. In our work we contributed by proposing SmoothDiff which is making use of the well-known mathematical structure of the Jacobian in neural networks together with automatic differentiation functionalities. Specifically, we show that the expected Jacobian can be decomposed efficiently into independent parts, with a focus on the relevant network non-linearities only. Automatic differentiation helps to readily implement this idea using vector-Jacobian products without further work. This results in a higher accuracy of the smoothed gradient convolution estimation at fewer samples: resulting in speedups of up to two orders of magnitude for multi-class explanations over the original SmoothGrad. This means that SmoothDiff helps to enhance the usability (speed and quality) of SmoothGrad greatly – as seen across all simulations and benchmarks explored. As smoothing comes with a reduction of curvature or second order derivatives, we conjecture that smoothed explanations as the ones studied here may also provide a higher resilience in the spirit of [23].

We can see three potentially interesting directions to further improve SmoothDiff: (i) Instead of approximating Gaussian convolutions by random samples, more efficient means of quadrature should be investigated. (ii) Moving beyond Gaussian filtering over the entire input domain, the geometry of the data manifold can be taken into account for convolutional smoothing, as shown in [48]. Insights from SmoothDiff could also be applied to other methods integrating over sensitivities, e.g. *Integrated Gradients* [44]. (iii) Instead of sampling activations on multiple forward passes, probability distributions could be propagated in a single forward pass, possibly further accelerating the method.

Future work will furthermore explore smoothed differentiation as in SmoothDiff beyond XAI, for example in applications of AI for the sciences where smooth priors in terms of differential equations exist, say, in the laws of physics.

Contributions

AH devised the project and main conceptual ideas. He wrote the majority of the manuscript, including experiments, benchmarks, figures, tables and code. NM contributed to the initial design of the method during his lab rotation under the guidance of AH. He evaluated the 1D example in Figure 1(a) and Appendix B, compared color schemes in Appendix D, wrote the initial draft of Appendix H, created the ReLU pseudocode listing in Appendix I, and plotted Figure 2 from AH's experimental results. JM wrote the PyTorch implementation of SmoothDiff, evaluated the *GridPG* experiments in Appendix F.2 and contributed to Section 2.4. SB advised the pixel-flipping experi-

ments in Section 3.4 and helped frame the paper. KRM supervised the project and helped frame, structure and write the paper. All authors discussed the results and commented on the manuscript.

Acknowledgements

We would like to thank the anonymous reviewers for their feedback, especially reviewer "Meht". We also thank Christopher Anders, Shinichi Nakajima, Anna Hedström, and Stefan Gugler for their valuable feedback and insightful discussions. Furthermore, we gratefully acknowledge funding from the German Federal Ministry of Education and Research under the grant BIFOLD25B. KRM was partly supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2019-II190079, Artificial Intelligence Graduate School Program, Korea University) and grant funded by the Korea government (MSIT) (No. RS-2024-00457882, AI Research Hub Project).

References

[1] W. Samek, G. Montavon, S. Lapuschkin, C. J. Anders, and K.-R. Müller, "Explaining Deep Neural Networks and Beyond: A Review of Methods and Applications," *Proceedings of the IEEE*, vol. 109, no. 3, pp. 247–278, Mar. 2021, doi: 10.1109/JPROC.2021.3060483.

[2] S. Lapuschkin, S. Wäldchen, A. Binder, G. Montavon, W. Samek, and K.-R. Müller, "Unmasking Clever Hans predictors and assessing what machines really learn," *Nature communications*, vol. 10, p. 1096, 2019.

[3] J. Kauffmann, J. Dippel, L. Ruff, W. Samek, K.-R. Müller, and G. Montavon, "Explainable AI reveals Clever Hans effects in unsupervised learning models," *Nature Machine Intelligence*, vol. 7, pp. 412–422, 2025.

[4] D. Baehrens, T. Schroeter, S. Harmeling, M. Kawanabe, K. Hansen, and K.-R. Müller, "How to Explain Individual Classification Decisions," *Journal of Machine Learning Research*, vol. 11, no. 61, pp. 1803–1831, 2010, http://jmlr.org/papers/v11/baehrens10a.html.

[5] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps." http://arxiv.org/abs/1312.6034

[6] D. Smilkov, N. Thorat, B. Kim, F. Viégas, and M. Wattenberg, "SmoothGrad: removing noise by adding noise." http://arxiv.org/abs/1706.03825

[7] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation," *PloS one*, vol. 10, no. 7, p. e130140, 2015.

[8] G. Montavon, S. Lapuschkin, A. Binder, W. Samek, and K.-R. Müller, "Explaining nonlinear classification decisions with deep Taylor decomposition," *Pattern Recognition*, vol. 65, pp. 211–222, May 2017, doi: 10.1016/j.patcog.2016.11.008.

[9] O. Eberle, J. Büttner, F. Kräutli, K.-R. Müller, M. Valleriani, and G. Montavon, "Building and interpreting deep similarity models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 3, pp. 1149–1161, 2022.

[10] T. Schnake *et al.*, "Higher-order explanations of graph neural networks via relevant walks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 44, no. 11, pp. 7581–7596, 2022.

[11] E. Štrumbelj and I. Kononenko, "An Efficient Explanation of Individual Classifications using Game Theory," *Journal of Machine Learning Research*, vol. 11, no. 1, pp. 1–18, 2010, http://jmlr.org/papers/v11/strumbelj10a.html.

[12] S. M. Lundberg and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions," in *Advances in Neural Information Processing Systems*, Curran Associates,

Inc., 2017. https://proceedings.neurips.cc/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html.

[13] D. Balduzzi, M. Frean, L. Leary, J. P. Lewis, K. W.-D. Ma, and B. McWilliams, "The Shattered Gradients Problem: If resnets are the answer, then what is the question?," in *Proceedings of the 34th International Conference on Machine Learning*, PMLR, Jul. 2017, pp. 342–350. https://proceedings.mlr.press/v70/balduzzi17b.html.

[14] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition." http://arxiv.org/abs/1409.1556

[15] M. Blondel and V. Roulet, "The Elements of Differentiable Programming." http://arxiv.org/abs/2403.14606

[16] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient BackProp," *Neural Networks: Tricks of the Trade: Second Edition*. Springer, Berlin, Heidelberg, pp. 9–48, 2012. doi: 10.1007/978-3-642-35289-8_3.

[17] A. Griewank and A. Walther, *Evaluating Derivatives*. in Other Titles in Applied Mathematics. Society for Industrial, Applied Mathematics, 2008. doi: 10.1137/1.9780898717761.

[18] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: a survey," *Journal of machine learning research*, vol. 18, no. 153, pp. 1–43, 2018.

[19] A. Paszke *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2019. https://proceedings.neurips.cc/paper_files/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html.

[20] J. Bradbury *et al.*, "JAX: composable transformations of Python+NumPy programs." http://github.com/google/jax

[21] K. Bykov, A. Hedström, S. Nakajima, and M. M.-C. Höhne, "NoiseGrad — Enhancing Explanations by Introducing Stochasticity to Model Weights," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 6, pp. 6132–6140, Jun. 2022, doi: 10.1609/aaai.v36i6.20561.

[22] S. Muzellec, T. Fel, V. Boutin, L. Andéol, R. Vanrullen, and T. Serre, "Saliency strikes back: How filtering out high frequencies improves white-box explanations," in *Proceedings of the 41st International Conference on Machine Learning*, PMLR, Jul. 2024, pp. 37041–37075. https://proceedings.mlr.press/v235/muzellec24a.html.

[23] A.-K. Dombrowski, M. Alber, C. Anders, M. Ackermann, K.-R. Müller, and P. Kessel, "Explanations can be manipulated and geometry is to blame," in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2019. https://papers.nips.cc/paper_files/paper/2019/hash/bb836c01cdc9120a9c984c525e4b1a4a-Abstract.html.

[24] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016, pp. 770–778. doi: 10.1109/CVPR.2016.90.

[25] A. H. Gebremedhin, F. Manne, and A. Pothen, "What Color Is Your Jacobian? Graph Coloring for Computing Derivatives," *SIAM Review*, vol. 47, no. 4, pp. 629–705, Jan. 2005, doi: 10/cmwds4.

[26] A. Hill and G. Dalle, "Sparser, Better, Faster, Stronger: Sparsity Detection for Efficient Automatic Differentiation," *Transactions on Machine Learning Research*, Jun. 2025, https://openreview.net/forum?id=GtXSN52nIW.

[27] A. Hill, G. Dalle, and A. Montoison, "An Illustrated Guide to Automatic Sparse Differentiation," in *The Fourth Blogpost Track at ICLR 2025*, 2025. https://openreview.net/forum?id=ykZibuSbJj.

[28] W. Baur and V. Strassen, “The complexity of partial derivatives,” *Theoretical Computer Science*, vol. 22, no. 3, pp. 317–330, Feb. 1983, doi: 10.1016/0304-3975(83)90110-X.

[29] J. Maeß, G. Montavon, S. Nakajima, K.-R. Müller, and T. Schnake, “Uncovering the Structure of Explanation Quality with Spectral Analysis.” http://arxiv.org/abs/2504.08553

[30] R. Achtibat *et al.*, “From attribution maps to human-understandable explanations through Concept Relevance Propagation,” *Nature Machine Intelligence*, vol. 5, no. 9, pp. 1006–1019, Sep. 2023, doi: 10.1038/s42256-023-00711-8.

[31] L. Linhardt, K.-R. Müller, and G. Montavon, “Preemptively pruning Clever-Hans strategies in deep neural networks,” *Information Fusion*, vol. 103, p. 102094, Mar. 2024, doi: 10.1016/j.inffus.2023.102094.

[32] A. Dosovitskiy *et al.*, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale,” in *International Conference on Learning Representations*, 2021. https://openreview.net/forum?id=YicbFdNTTy.

[33] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A large-scale hierarchical image database,” in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2009, pp. 248–255. doi: 10.1109/CVPR.2009.5206848.

[34] A. Hill, “Julia-XAI/VisionHeatmaps.jl: v2.0.1.” https://zenodo.org/records/15021377

[35] A. Hill, “ExplainableAI.jl.” https://zenodo.org/records/15176551

[36] M. Innes *et al.*, “Fashionable Modelling with Flux,” *CoRR*, 2018, https://arxiv.org/abs/1811.01457.

[37] M. Innes, “Flux: Elegant machine learning with Julia,” *Journal of Open Source Software*, vol. 3, no. 25, p. 602, May 2018, doi: 10.21105/joss.00602.

[38] F. White *et al.*, “JuliaDiff/ChainRules.jl: v1.72.3.” https://zenodo.org/records/14926720

[39] M. Innes, “Don't Unroll Adjoint: Differentiating SSA-Form Programs,” *CoRR*, 2018, http://arxiv.org/abs/1810.07951.

[40] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, “Julia: A fresh approach to numerical computing,” *SIAM review*, vol. 59, no. 1, pp. 65–98, 2017, https://doi.org/10.1137/141000671.

[41] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, “Image quality assessment: from error visibility to structural similarity,” *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, Apr. 2004, doi: 10.1109/TIP.2003.819861.

[42] S. Bluecher, J. Vielhaben, and N. Strodthoff, “Decoupling Pixel Flipping and Occlusion Strategy for Consistent XAI Benchmarks,” *Transactions on Machine Learning Research*, Mar. 2024, https://openreview.net/forum?id=bIiLXdtUVM.

[43] S. Hooker, D. Erhan, P.-J. Kindermans, and B. Kim, “A Benchmark for Interpretability Methods in Deep Neural Networks,” in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2019. https://papers.nips.cc/paper_files/paper/2019/hash/fe4b8556000d0f0cae99daa5c5c5a410-Abstract.html.

[44] M. Sundararajan, A. Taly, and Q. Yan, “Axiomatic attribution for deep networks,” in *International conference on machine learning*, PMLR, 2017, pp. 3319–3328.

[45] G. Montavon, A. Binder, S. Lapuschkin, W. Samek, and K.-R. Müller, “Layer-wise relevance propagation: an overview,” *Explainable AI: interpreting, explaining and visualizing deep learning*, pp. 193–209, 2019.

[46] M. Kohlbrenner, A. Bauer, S. Nakajima, A. Binder, W. Samek, and S. Lapuschkin, “Towards Best Practice in Explaining Neural Network Decisions with LRP,” in *2020 International Joint Conference on Neural Networks (IJCNN)*, Jul. 2020, pp. 1–7. doi: 10.1109/IJCNN48605.2020.9206975.

[47] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 618–626.

[48] L. Zhou, C. Ma, Z. Wang, and X. Shi, "Rethinking the Principle of Gradient Smooth Methods in Model Explanation." http://arxiv.org/abs/2410.07711

[49] L. Arras, A. Osman, and W. Samek, "CLEVR-XAI: A benchmark dataset for the ground truth evaluation of neural network explanations," *Information Fusion*, vol. 81, pp. 14–40, May 2022, doi: 10.1016/j.inffus.2021.11.008.

[50] F. Crameri, G. E. Shephard, and P. J. Heron, "The misuse of colour in science communication," *Nature Communications*, vol. 11, no. 1, p. 5444, Oct. 2020, doi: 10.1038/s41467-020-19160-7.

[51] F. Crameri, "Scientific colour maps." https://zenodo.org/records/8409685

[52] P. Kovesi, "Good Colour Maps: How to Design Them," *arXiv:1509.03700 [cs]*, Sep. 2015, http://arxiv.org/abs/1509.03700.

[53] R. Bujack, T. L. Turton, F. Samsel, C. Ware, D. H. Rogers, and J. Ahrens, "The Good, the Bad, and the Ugly: A Theoretical Framework for the Assessment of Continuous Colormaps," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 1, pp. 923–933, Jan. 2018, doi: 10.1109/TVCG.2017.2743978.

[54] M. R. Luo, G. Cui, and B. Rigg, "The development of the CIE 2000 colour-difference formula: CIEDE2000," *Color Research & Application*, vol. 26, no. 5, pp. 340–350, 2001, doi: 10.1002/col.1049.

[55] A. Hedström *et al.*, "Quantus: An Explainable AI Toolkit for Responsible Evaluation of Neural Network Explanations and Beyond," *Journal of Machine Learning Research*, vol. 24, no. 34, pp. 1–11, 2023, http://jmlr.org/papers/v24/22-0142.html.

[56] S. Gao, Z.-Y. Li, M.-H. Yang, M.-M. Cheng, J. Han, and P. Torr, "Large-scale Unsupervised Semantic Segmentation," *TPAMI*, 2022.

[57] N. Kokhlikyan *et al.*, "Captum: A unified and generic model interpretability library for pytorch," *arXiv preprint arXiv:2009.07896*, 2020.

[58] C. J. Anders, D. Neumann, W. Samek, K.-R. Müller, and S. Lapuschkin, "Software for dataset-wide XAI: from local explanations to global insights with Zennit, CoRelAy, and ViRelAy," *arXiv preprint arXiv:2106.13200*, 2021.

[59] D. Alvarez-Melis and T. S. Jaakkola, "On the Robustness of Interpretability Methods." http://arxiv.org/abs/1806.08049

[60] D. Alvarez Melis and T. Jaakkola, "Towards Robust Interpretability with Self-Explaining Neural Networks," in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2018. https://proceedings.neurips.cc/paper_files/paper/2018/hash/3e9f0fc9b2f89e043bc6233994dfcf76-Abstract.html.

[61] C.-K. Yeh, C.-Y. Hsieh, A. Suggala, D. I. Inouye, and P. K. Ravikumar, "On the (in) fidelity and sensitivity of explanations," *Advances in neural information processing systems*, vol. 32, 2019.

[62] M. Böhle, M. Fritz, and B. Schiele, "Convolutional Dynamic Alignment Networks for Interpretable Classifications," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2021, pp. 10024–10033. doi: 10.1109/CVPR46437.2021.00990.

[63] M. Böhle, N. Singh, M. Fritz, and B. Schiele, "B-cos alignment for inherently interpretable CNNs and vision transformers," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, no. 6, pp. 4504–4518, 2024.

[64] U. Bhatt, A. Weller, and J. M. F. Moura, "Evaluating and Aggregating Feature-based Model Explanations," in *Proceedings of the Twenty-Ninth International Joint Conference on Artifi-*

cial Intelligence, Yokohama, Japan: International Joint Conferences on Artificial Intelligence Organization, Jul. 2020, pp. 3016–3022. doi: 10.24963/ijcai.2020/417.

[65] P. Chalasani, J. Chen, A. R. Chowdhury, X. Wu, and S. Jha, "Concise Explanations of Neural Networks using Adversarial Training," in *Proceedings of the 37th International Conference on Machine Learning*, PMLR, Nov. 2020, pp. 1383–1391. https://proceedings.mlr.press/v119/chalasani20a.html.

[66] G. Montúfar, R. Pascanu, K. Cho, and Y. Bengio, "On the Number of Linear Regions of Deep Neural Networks," in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2014. https://proceedings.neurips.cc/paper_files/paper/2014/hash/fa6f2a469cc4d61a92d96e74617c3d2a-Abstract.html.

[67] M. Ancona, E. Ceolini, C. Öztireli, and M. Gross, "Towards better understanding of gradient-based attribution methods for Deep Neural Networks," Feb. 2018. https://openreview.net/forum?id=Sy21R9JAW.

[68] A.-K. Dombrowski, J. E. Gerken, and P. Kessel, "Diffeomorphic Explanations with Normalizing Flows," in *ICML Workshop on Invertible Neural Networks, Normalizing Flows, and Explicit Likelihood Models*, 2021. https://openreview.net/forum?id=ZBR9EpEl6G4.

A Notation

We denote scalar quantities by lowercase letters x, vector quantities by bold lowercase letters $\boldsymbol{x}$ and matrices by bold uppercase letters $\boldsymbol{X}$. The coefficients in a vector $\boldsymbol{x}$ are written as x_i, the rows of a matrix $\boldsymbol{X}$ as $\boldsymbol{X}_{i,:}$ and the coefficients as $[\boldsymbol{X}]_{i,j} = X_{i,j}$, using row and column indices respectively. All vectors $\boldsymbol{x}$ are column-vectors and all $\boldsymbol{x}^T$ are row-vectors. For a vector-to-scalar function $f : \mathbb{R}^n \to \mathbb{R}$, we use $\nabla f(\boldsymbol{x}) \in \mathbb{R}^n$ for the gradient vector. For a vector-to-vector function $f : \mathbb{R}^n \to \mathbb{R}^m$, we denote as $\boldsymbol{J}_f(\boldsymbol{x}) \in \mathbb{R}^{n \times m}$ the Jacobian matrix of first-order partial derivatives $\frac{\partial f_i(\boldsymbol{x})}{\partial x_j}$. The i-th basis vector is written as $\boldsymbol{e}_i$. We reserve the index c for the selection of the output neurons, such that for a neural network f and an input $\boldsymbol{x}$, the predicted logit for the c-th class corresponds to $f_c(\boldsymbol{x})$.

B Experiment details – Figure 1

(a) 1-dimensional toy example: To demonstrate the SGP on gradient-based XAI methods in Figure 1 (a), we define a deep neural network similar to [13]. We use a sequence of 16 fully connected ReLU layers with 64 hidden neurons each, mapping scalar inputs to scalar outputs. Both weights and biases are drawn from a normal distribution with mean zero and variance $\frac{1}{N}$. Both SmoothDiff and SG explanations are computed using $\sigma = 0.05$ and $n = 10$ samples. Since the explanations are stochastic (inputs are randomly sampled), we visualize the means and 5th- to 95th-percentile bands over 1024 evaluations per input. β-Smoothing was computed using $\beta = \log(2)\sqrt{\frac{2\pi}{\sigma}}L \approx 117$, motivated by scaling up the analytical derivation for a single layer [23]. The Gaussian convolution was computed numerically from the gradient.

(b) Vision model: Explanations are computed on a pretrained VGG-19 model [14] and randomly selected images from the ImageNet dataset [33]. They are visualized using the heatmapping techniques described in Appendix D. Both SmoothDiff and SG are computed using $\sigma = 0.5$. Since the analytical Gaussian convolution is intractable and an accurate numerical computation is infeasible, it was approximated by computing SG with $n = 10^6$ samples (see Section 3.1).

C Relevance pooling

Most XAI methods, including SmoothDiff, have in common that the dimensionality of their numerical explanations matches that of the input features. The inputs to ImageNet models are (normalized) RGB images: three-dimensional tensors that include width, height and color channel dimensions. Explanations of these inputs therefore also contain three separate values per pixel, one for each color channel (red, green, blue). For the purpose of heatmapping (see Appendix D) as well as several evaluation metrics (see Appendix F), these three values need to be reduced to a single scalar value by applying a *relevance pooling* function $R_{\text{pool}} : \mathbb{R}^3 \to \mathbb{R}$ to each pixel [49]. Popular options include summation, max pooling and the ℓ^2-norm (hereafter referred to as norm-pooling).

As motivated in Section 1, explanations can be based on different mathematical objects, e.g. gradients, Taylor expansions and Shapley values. The choice of relevance pooling depends on the XAI method: For LRP, summation is the most common choice, as it maintains its property of *relevance conservation* [7]. Sum-pooled explanations can contain both positive and negative values and are therefore commonly visualized using diverging color maps, see e.g. LRP's characteristic red-white-blue heatmaps.

For gradient-based methods however, summation is not an option. We want to illustrate this with a thought experiment: When using a gradient-based method to explain why an image of a red fire truck is correctly classified by an idealized robust model, we might expect the gradient to be positive on the red color channel (a more intense red positively affecting the classification) and negative on the blue and green color channels (also intensifying the red color). In this example, negative sensitivities on the blue and green color channels do not imply that the pixel is of negative importance to the classifier. Sum-pooling should therefore be avoided, as it can in the worst case result in a negative value even though the pixel was of high sensitivity. Instead, we suggest the use of the norm-pooling, measuring the pixel-wise gradient magnitude. Since norm-pooled explanations are non-negative,

they should be visualized with sequential color maps (see Appendix D). As we will demonstrate in Appendix F, the choice of relevance pooling has a significant impact on the evaluation of XAI methods, a crucial aspect that remains underexplored in the literature.

D Heatmapping

We visualize numerical explanations as heatmaps using the *VisionHeatmaps.jl* package [34] from the *Julia-XAI* ecosystem [35]. To map gradient-based explanations onto a color scheme, we reduce the color channel dimension by applying norm-pooling (see Appendix C). Finally, to avoid desaturation due to numerical outliers, the color scheme is mapped to a range between 0 and the 99.9th percentile of the pooled explanation, effectively normalizing values within the explanation.

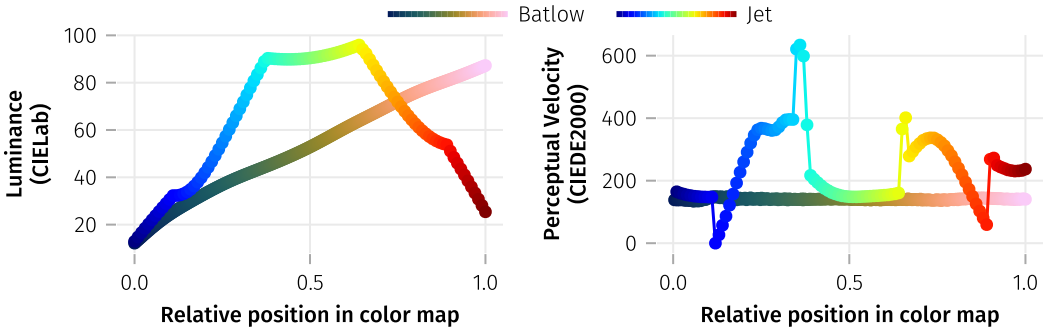

Figure 5: Comparison of perceptual properties of the `jet` and `batlow` color maps.

We use the perceptually uniform `batlow` color map [50, 51]. For sequential color maps, a linear increase in luminance is desirable, as it is the primary perceptual dimension perceived as intensity, as well as the one best resolved at high frequencies [52]. Additionally, constant change in overall perceptual difference between adjacent colors ensures uniformity [53]. Figure 5 compares `batlow` with the popular `jet` color map, measuring luminance L^* in the *CIELab* color space and overall perceptual velocity using the *CIEDE2000* distance metric [54]. While we discourage the use of the `jet` color map due to its perceptual non-uniformity, especially in combination with highly processed transparent overlays, we provide such SmoothDiff heatmaps in Figure 6 due to their ongoing popularity in the field.

Figure 6: Comparison of heatmapping techniques on SmoothDiff explanations ($n = 10$, $\sigma = 0.5$).

E Characterizing neglected cross-covariances

As discussed in Section 3.1, SmoothDiff trades a small inherent bias in form of neglected cross-covariance terms in (5) for a large increase in sample efficiency. To quantify the bias, we modify the convergence experiment from Section 3.1, treating SmoothDiff and SG explanations (i.e. smoothed gradients) as vectors. We then measure their similarity to the converged SG reference explanation by computing cosine similarities and magnitudes.

The resulting convergence plots are shown Figure 7: In terms of cosine similarity, SmoothDiff shows superior convergence behavior up to SG's recommended sample size of $n = 50$. This indicates that the direction of SmoothDiff explanations aligns well with the reference. In terms of magnitude however, SmoothDiff has a clear bias: While SG converges towards the magnitude of the reference (i.e. a relative magnitude of one), SmoothDiff converges towards a much smaller relative magnitude of 0.461. We suspect that this difference in magnitude is caused by the neglected cross-covariances.

As described in Appendix D, explanations are relevance pooled and normalized during heatmapping. Heatmaps therefore only depend on the direction of the explanation vector, not the magnitude, limiting the effect of neglected cross-covariances. This observation is in line with the SSIM convergence plot in Figure 2.

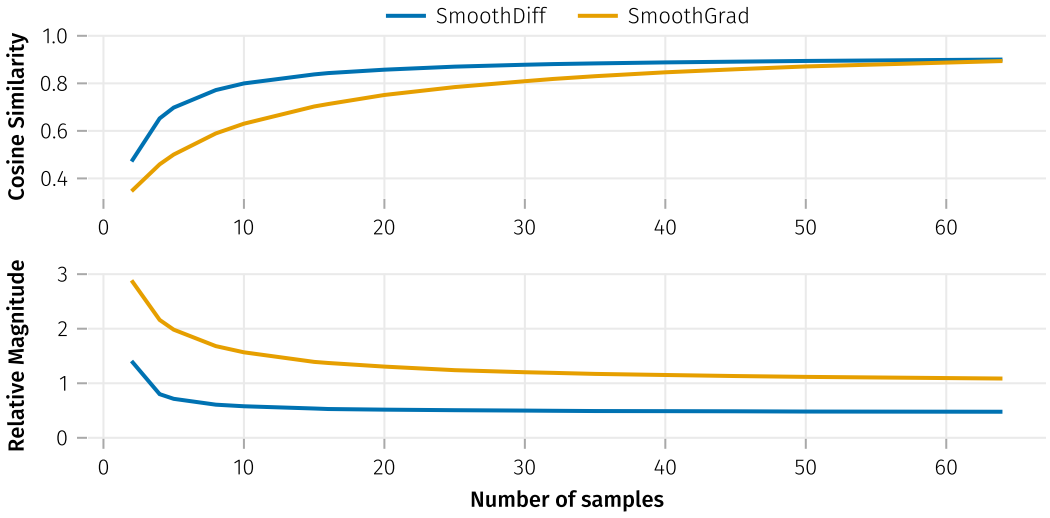

Figure 7: Convergence behavior of SmoothDiff and SG towards the reference explanation in terms of cosine similarity (upper) and relative magnitude of the explanation (lower). Convergence with the reference is obtained at a cosine similarity and relative magnitude of one.

F Further quantitative results

In addition to the pixel-flipping experiments which evaluate *faithfulness* in Section 3.4, we also evaluate metrics in the domains of *robustness*, *localization* and *complexity* using the *Quantus* toolkit [55]. Since some localization metrics require ground truth segmentation masks that localize the object of the target class, a dataset of 256 input images, labels and segmentation masks was created by sub-sampling correctly classified images from the ImageNet-S-50 dataset [56].

In addition to the Gradient, SmoothDiff and SG methods in Section 3, we also evaluate *SmoothGrad-Squared* [43], *Integrated Gradients* [44], *GradCAM* [47], two *LRP composites* [45, 46], and three random baselines. Reference implementations from the *Captum* software package [57] are used for Integrated Gradients and GradCAM, while *Zennit* [58] is used for LRP, namely the `EpsilonPlus` and `EpsilonAlpha2Beta1` composites. Metrics are evaluated on pre-trained VGG-19 and ResNet-18 models using Quantus' default parameters.

As motivated in Appendix C, relevance pooling significantly impacts both heatmapping and evaluation metrics. Since the goal of XAI is to further humans' understanding of machine learning models,

we argue that evaluation metrics should apply the same relevance pooling functions that are used to compute heatmapping visualizations. We therefore evaluate gradient-based methods (Gradient, SmoothGrad, SmoothDiff, Integrated Gradients) with norm-pooling and methods based on Taylor expansions (LRP composites, Input x Gradient) with sum-pooling.

In [43], a variant of SG called SmoothGrad-Squared (SG-SQ) is shown to perform well using sum-pooling. Instead of averaging gradients, SG-SQ averages squared gradients.[6] While not exactly analogous,[7] we introduce SmoothDiff-Squared (SD-SQ), which we define as the square of a SmoothDiff explanation

$$[\text{SD-SQ}(\boldsymbol{x})]_i = [\text{SD}(\boldsymbol{x})]_i^2 \,. \tag{10}$$

We want to emphasize that a sum-pooled SD-SQ explanation is the square of a norm-pooled SmoothDiff explanation, blurring the lines between XAI method and relevance pooling. To further quantify the influence of relevance pooling on evaluation metrics, we introduce three random baselines that are equally uninformative:

- **Random (ℓ^2):** random explanation drawn from a normal distribution, norm-pooled
- **Random (Σ):** random explanation drawn from a normal distribution, sum-pooled
- **Random-Squared (Σ):** squared random explanation drawn from a normal distribution, sum-pooled

For SmoothDiff, SmoothDiff-Squared, SmoothGrad, SmoothGrad-Squared, and Integrated Gradients, explanations are computed using 20 samples in an attempt to make them computationally comparable. All evaluation metrics are evaluated on batches of 256 images described above. We visualize the resulting distributions of scores as box plots in Figures 8 to 13, additionally reporting means in the right-hand column. XAI methods and random baselines are purposefully arranged in three groups: gradient-based methods using norm-pooling, squared methods with sum-pooling (SG-SQ, SD-SQ, Random-Squared) and other sum-pooled methods.

F.1 Robustness

Robustness metrics evaluate how stable explanations are under slight input perturbations. We evaluate two robustness metrics, the first being the *Local Lipschitz Estimate* (LLE), described in [59, 60]. The resulting distribution of scores is visualized in Figure 8, where lower scores indicate higher robustness. On VGG-19, SmoothDiff-Squared outperforms all other methods by a significant margin. Results on ResNet-18 qualitatively mirror those of VGG-19, indicating that SmoothDiff and SmoothDiff-Squared explanations are robust across model architectures. LRP `EpsilonAlpha2Beta1` does however significantly improve on ResNet-18, achieving a slightly lower mean than SmoothDiff-Squared. Within gradient-based methods, SmoothDiff is the most robust on both models. The three random baselines reveal that the LLE is biased towards Random-Squared,[8] mirroring the fact that SmoothDiff-Squared outperforms SmoothDiff.

The second evaluated robustness metric is *Average Sensitivity* [61], shown in Figure 9. Once again, lower scores indicate higher robutstness. SmoothDiff's robustness on both VGG-19 and ResNet-18 is only outperformed by the two LRP composites. Within gradient-based methods, SmoothDiff again is the most robust method across model architectures. The random baselines reveal that the metric is biased towards norm-pooling, mirroring the fact that SmoothDiff outperforms SmoothDiff-Squared.

F.2 Localization

Localization metrics test whether the highest values of an explanation is located within a desired region of interest. In case of the first metric, *Relevance Rank Accuracy* (RRA) [49], the region of interest is specified by providing ground-truth segmentation masks, as described above. The resulting RRA scores are shown in Figure 10, where higher scores are desirable. In the three random baselines,

[6] When talking about squared gradients and explanations, we refer to the element-wise square.

[7] In terms of (1), SG-SQ approximates the expected value of the squared gradients while SD-SQ approximates the square of the expected value of gradients.

[8] Since all random baselines are equally informative, we would expect them to be equally robust.

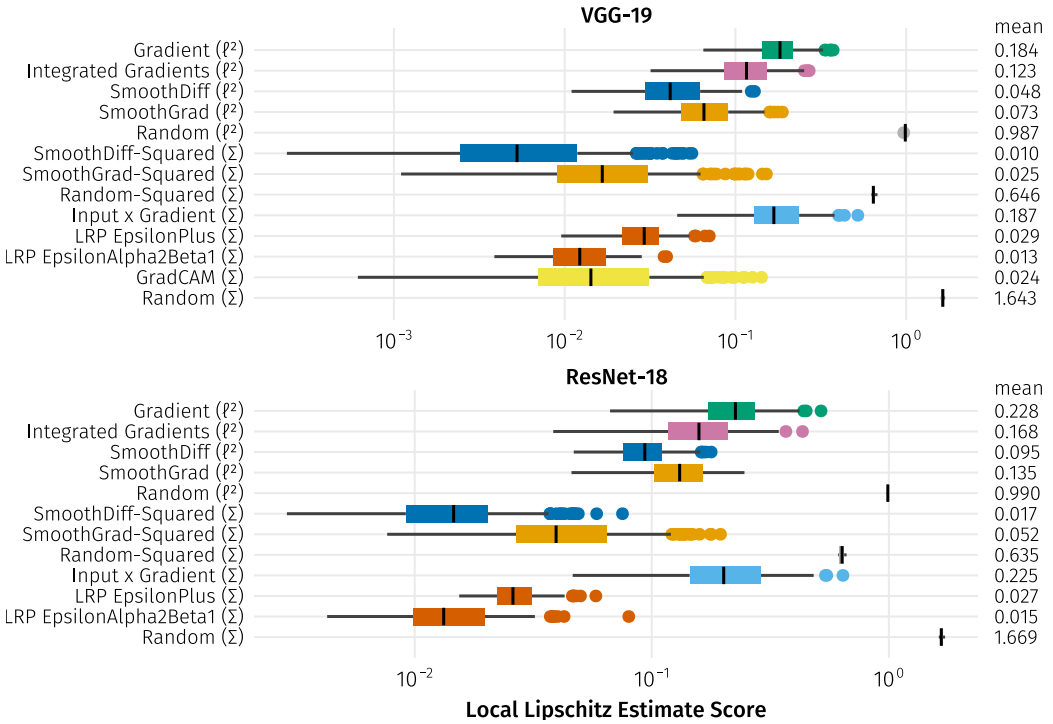

Figure 8: Evaluation of the *Local Lipschitz Estimate* [59, 60] as a measure of robustness. Lower scores indicate a higher robustness. We indicate relevance pooling functions next to XAI method names, using ℓ^2 for norm-pooling and Σ for sum-pooling.

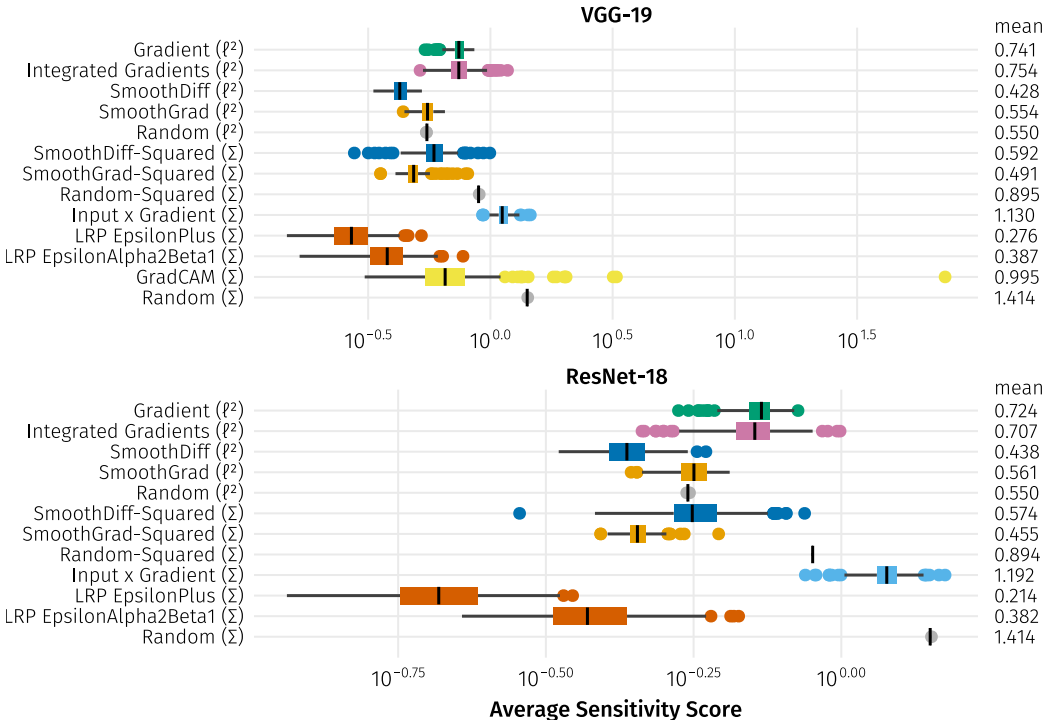

Figure 9: Evaluation of *Average Sensitivity* [61] as a measure of robustness. Lower scores indicate a higher robustness. We indicate relevance pooling functions next to XAI method names, using ℓ^2 for norm-pooling and Σ for sum-pooling.

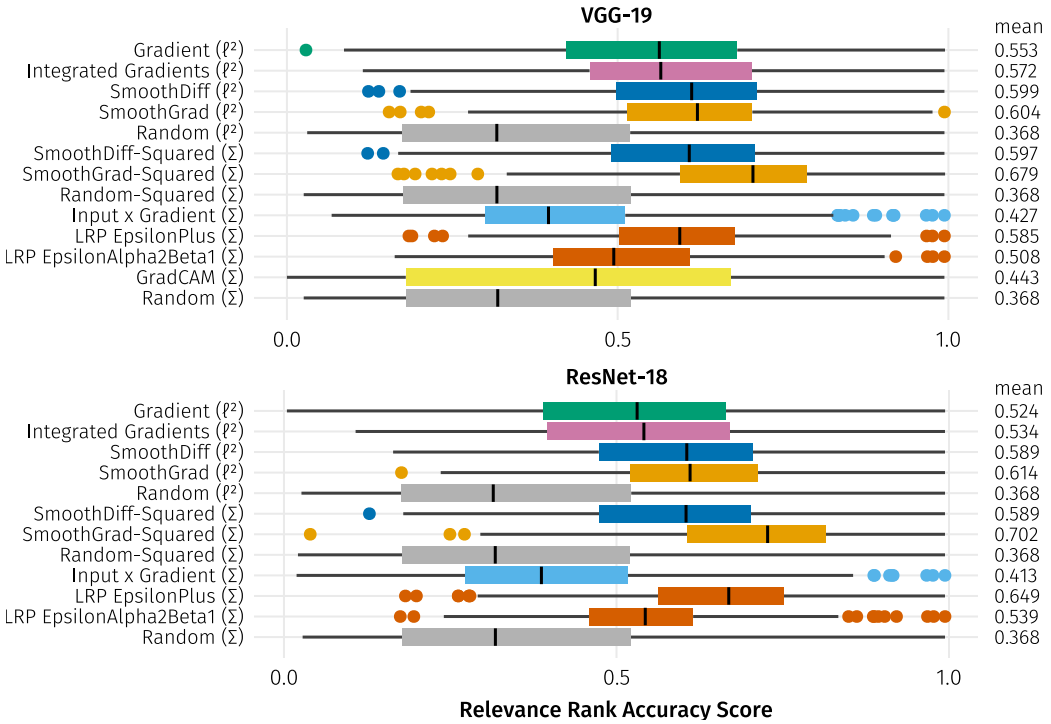

Figure 10: Evaluation the *Relevance Rank Accuracy* [49] as a measure of localization. Higher scores indicate better localization. We indicate relevance pooling functions next to XAI method names, using ℓ^2 for norm-pooling and Σ for sum-pooling.

the RRA shows no visible bias towards any of the evaluated relevance-pooling functions. SG-SQ performs best on both VGG-19 and ResNet-18, followed by SG, SmoothDiff and SmoothDiff-Squared, which all perform similarly well.

The second localization metric is the *Grid Pointing Game* (GridPG) [62, 63], an artificial benchmark in which four images from different classes are tiled in a 2×2 grid. The metric selects one of the images as the region of interest and target class of the explanation. Scores correspond to the sum of positive values within the target quadrant, divided by the sum of all positive values, normalizing scores between zero and one. The results are shown in Figure 11, where higher scores indicate better localization. Since explanations from all three random baselines are distributed equally across all four quadrants, all baselines obtain a score of 0.25. Due to its normalization, the GridPG appears to favor XAI methods (and corresponding relevance pooling functions) like LRP that can return negative values.[9] Since this is neither the case for gradient-based methods, nor for squared methods, these perform significantly worse than LRP composites and GradCAM.

F.3 Complexity

The final category of evaluation metrics quantifies the conciseness of explanations, favoring highlighting few features. The first metric is *Complexity* [64], shown in Figure 12. On VGG-19, GradCAM performs best, followed by SmoothDiff-Squared. On ResNet-18, SmoothDiff-Squared performs best (in terms of mean score) as Captum's GradCAM implementation could not be evaluated on the model. The random baselines reveal a bias for Random-Squared and the sum-pooled random explanation, mirroring the fact that SmoothDiff-Squared outperforms SmoothDiff.

The second and final complexity metric is *Sparseness* [65], which is based on the Gini Index. Results are shown in Figure 13. On VGG-19, they qualitatively mirror those of the complexity metric in Figure 12: GradCAM performs best, followed by SmoothDiff-Squared and LRP `EpsilonAlpha2Beta1`. On ResNet-18, SmoothDiff-Squared is a close second behind the LRP `EpsilonAlpha2Beta1`.

[9] Non-negative explanations will result in a larger denominator.

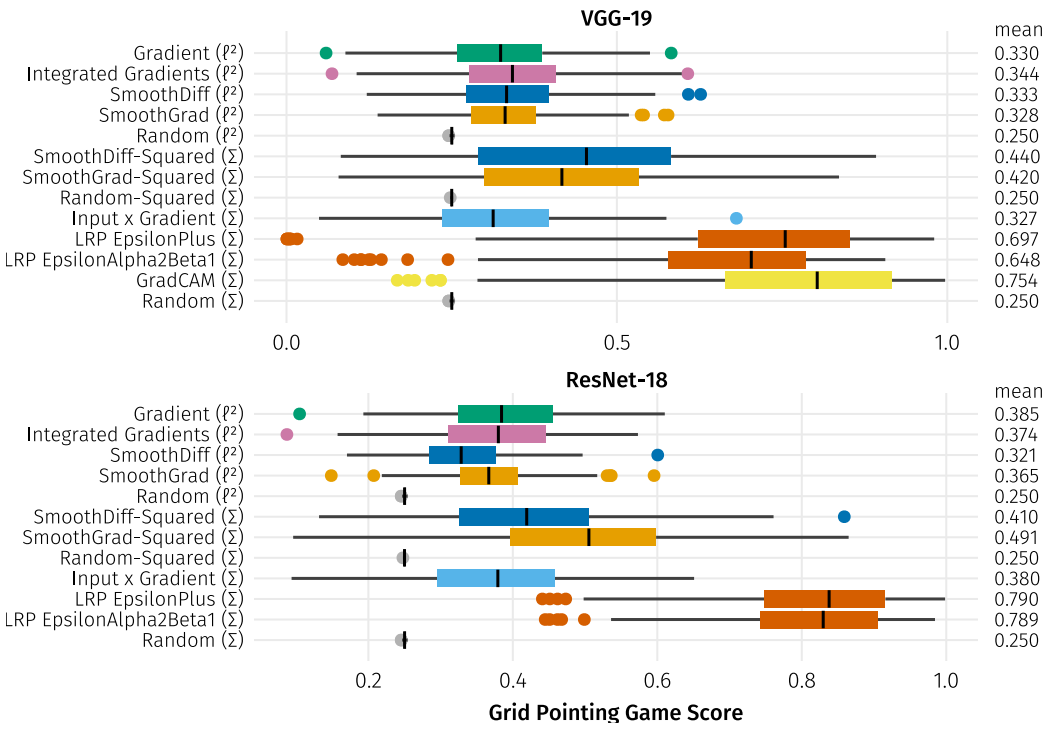

Figure 11: Evaluation of the *Grid Pointing Game* [62, 63] as a measure of localization. Higher scores indicate better localization. We indicate relevance pooling functions next to XAI method names, using ℓ^2 for norm-pooling and Σ for sum-pooling.

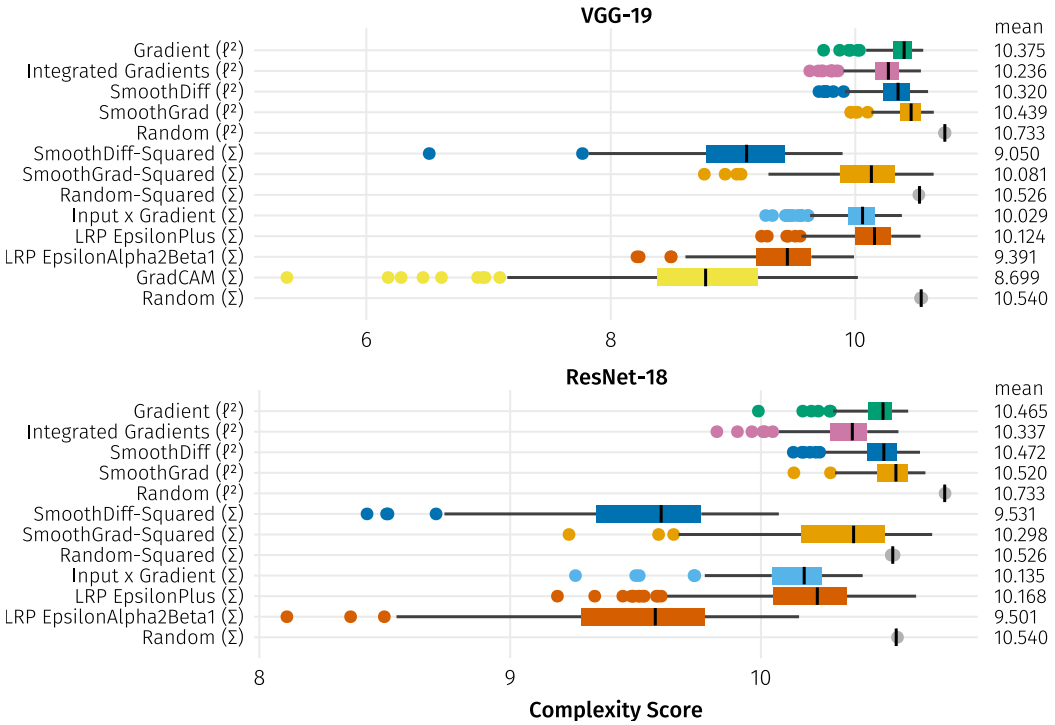

Figure 12: Evaluation of *Complexity* [64]. Lower scores indicate a better complexity. We indicate relevance pooling functions next to XAI method names, using ℓ^2 for norm-pooling and Σ for sum-pooling.

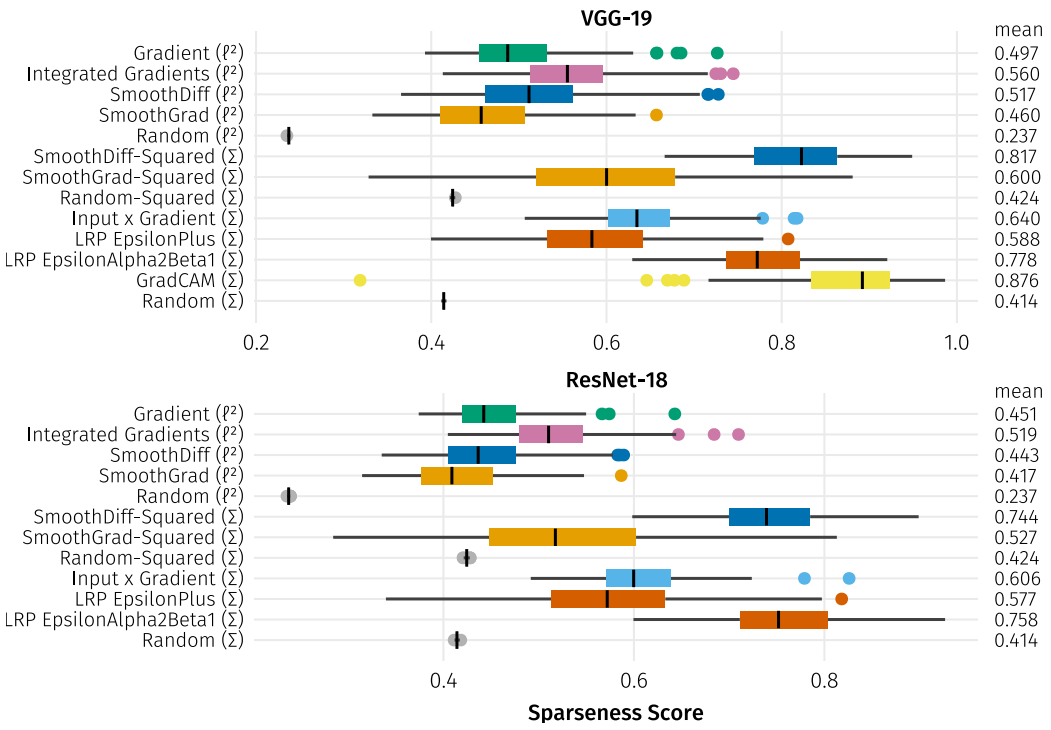

Figure 13: Evaluation of *Sparseness* [65] as a measure of complexity. Higher scores indicate a better complexity. We indicate relevance pooling functions next to XAI method names, using ℓ^2 for norm-pooling and Σ for sum-pooling.

The metric shows a significant bias against the norm-pooled random explanation, which can also be seen in the comparatively low scores of all gradient-based explanations.

In conclusion, SmoothDiff and SmoothDiff-Squared perform very well in terms of localization, robustness and complexity, three desired properties in XAI, while being performant and simple to implement. Using three equally uninformative random baselines (which should have performed equally well in terms of robustness, localization and complexity), we demonstrated that relevance pooling functions strongly alter the scores of most metrics, highlighting their significant (but underexplored) impact on the evaluation and development of XAI methods, as well as the resulting difficulty of comparing methods that require different relevance pooling functions.

G β-Smoothing and Gaussian convolutions

In [15], the analytical solution for the convolution of a ReLU activation function r with a Gaussian kernel g_σ is shown to be

$$(r * g_\sigma)(x) = \underbrace{\frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}\sigma}\right)\right]}_{\Phi_\sigma(x)} x + \sigma^2 g_\sigma(x) \tag{11}$$

with derivative $(r * g_\sigma)'(x) = \Phi_\sigma(x)$, where erf is the error function. The authors term these functions *smoothed ReLU* and *smoothed Heaviside* respectively. For a single-layer neural network, SmoothDiff's expected ReLU Jacobians (8) are the sampling approximation to the smoothed Heaviside function.

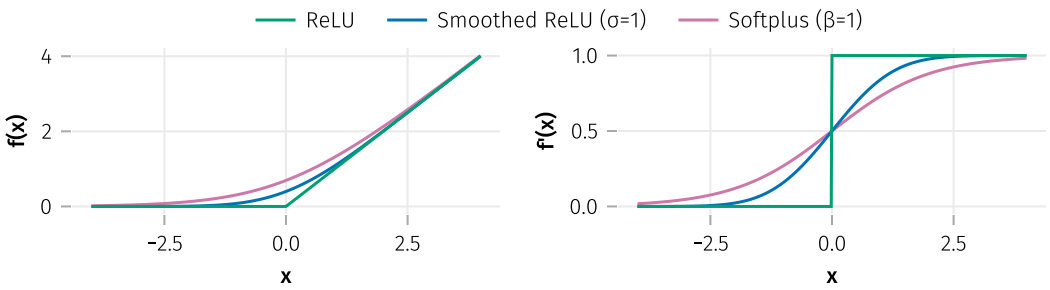

Figure 14: Comparison of ReLU, smoothed ReLU, and softplus functions (left) and their derivatives (right).

In [23], it is shown that gradient-based explanations on deep ReLU networks can be manipulated with hardly perceptible perturbations. The authors argue that this vulnerability is due to the large curvature of such networks and propose β-Smoothing as a robust explanation method, replacing ReLU activation functions by softplus activations

$$\text{softplus}_\beta(x) = \frac{1}{\beta}\log\left(1 + e^{\beta x}\right) \tag{12}$$

with small parameters β around one (matching pixel-flipping results in Table 1, in which $\beta = 1$ performed best).

The authors demonstrate that for single-layer neural networks, β-Smoothing can be understood as an approximation to the infinite sample limit of SG and therefore an approximation to the Gaussian convolution (1). Figure 14 compares ReLU, smoothed ReLU and softplus functions, as well as their derivatives. β-Smoothing provides a good approximation of the smoothed ReLU and can be further improved by tuning the β parameter.

While Gaussian convolutions are well approximated by β-Smoothing for single-layer networks, non-linear functions break the assumption of Gaussian activations in subsequent layers. Resulting heatmaps for β-Smoothing are shown in Figure 17. While the method qualitatively improves upon regular gradients, it results in heatmaps that approximate neither SG nor SmoothDiff and are more noisy and less contoured.

H SmoothDiff directly addresses the shattered gradient problem

Deep ReLU networks are piece-wise linear functions. The vector field of gradients of such networks is piece-wise constant and discontinuous, as shown in Figure 1 (a). In [66], it is shown that with increasing network depth, the number of linear regions and therefore discontinuities grows exponentially. These discontinuities are the source of the SGP for deep ReLU networks, which can be demonstrated using a path formulation of the gradient [13]. In Appendix H.1, we introduce our notation for a general path formulation similar to that of [67]. We apply it to deep ReLU networks in Appendix H.2, and demonstrate how SmoothDiff directly addresses it in Appendix H.3. We then extend the same formulation to general non-linearities in Appendix H.4.

H.1 Notation

Given a neural network's directed computational graph, we denote by $P_{i\to c}$ the set of all paths p from input i to output c. Applying this notation to (3), the i-th entry of the gradient $\nabla f_c(\boldsymbol{x})$ can be written as

$$\left[\nabla f_c(\boldsymbol{x})\right]_i = \frac{\partial f_c(\boldsymbol{x})}{\partial x_i} = \sum_{p\in P_{i\to c}} \prod_n \left[\boldsymbol{J}_{f^n}(\boldsymbol{a}^n)\right]_p, \tag{13}$$

where $\left[\boldsymbol{J}_{f^n}\right]_p$ are scalar Jacobian entries of functions f^n along the path.

We further factorize this product into:

- Linear or affine functions, whose Jacobian entries $\left[\boldsymbol{J}_{f^n}\right]_p = W_p^n$ are constant (for dense and convolutional layers, these correspond to individual weights).
- Non-linear functions, whose Jacobians entries $\left[\boldsymbol{J}_{f^n}(\boldsymbol{a}^n)\right]_p = g_p^n(\boldsymbol{a}^n)$ depend on one or more values in the input activation $\boldsymbol{a}^n$ (which in turn depend on $\boldsymbol{x}$, see Section 1).

This results in the path formulation

$$\frac{\partial f_c(\boldsymbol{x})}{\partial x_i} = \sum_{p \in P_{i \to c}} \underbrace{\prod_i W_p^i}_{=:W_p} \underbrace{\prod_j g_p^j(\boldsymbol{a}^j)}_{=:G_p(\boldsymbol{x})} = \sum_{p \in P_{i \to c}} W_p \cdot G_p(\boldsymbol{x}) \,. \tag{14}$$

Analogous to [13], we refer to the product W_p as the *path-weight*. Note that since all Jacobian entries in (13) are scalar, products commute and the order of functions in the computational graph doesn't affect their contribution to $\partial f_c / \partial x_i$. Equation (14) therefore generalizes to both arbitrary compositional structures and arbitrary non-linearities.

H.2 Deep ReLU networks

We now apply the path formulation of the gradient (14) to deep ReLU networks, which exclusively contain ReLU and max pooling non-linearities. For such networks, terms g_p^j are indicator functions and evaluate to either one or zero (see Section 2.2). We denote a path p as *active* if its product G_p is non-zero. This is the case if and only if the path contains the largest inputs into every max pooling filter and all inputs to ReLUs on the path are positive. If a path is active, it contributes the path-weight W_p to the gradient.

Since changes in the input $\boldsymbol{x}$ alter all activations $\boldsymbol{a}^j$, they also affect terms $g_p^j(\boldsymbol{a}^j)$. If even a single of these terms turns zero (e.g. because an activation into a ReLU becomes negative), contributions from the path-weight W_p vanish. Reversely, changes in the input can also activate new paths, adding their respective path-weight W_p. With deeper networks, the length of paths increases linearly and their number exponentially, increasing the density of path activity changes over the input domain. As a result, small changes in the input cause more and more frequent changes in the gradient, eventually resembling white noise [13]. We can express this sensitivity of paths by the second-order partial derivative

$$\frac{\partial^2 f_c(\boldsymbol{x})}{\partial x_i \partial x_j} = \sum_{p \in P_{i \to c}} W_p \cdot \frac{\partial G_p(\boldsymbol{x})}{\partial x_j} \,. \tag{15}$$

Since G_p are indicator functions, their derivatives at points of path activity change are infinite (or undefined). The sensitivity of paths in (15) directly depends on the curvature of f_c. Since f_c is a piece-wise linear function, its curvature is either zero within a linear region or infinite (or undefined) at junction points of regions.

H.3 Addressing shattered gradients

As described in Section 1, applying Gaussian convolutions (1) to piece-wise constant gradients $\nabla f_c(\boldsymbol{x})$ results in smooth continuous functions, introducing curvature (see Figure 1 a). Even though it doesn't directly act on the mechanisms described in Appendix H.2, SG therefore addresses the shattered gradient problem by approximating said Gaussian convolution in the infinite sample limit.

SmoothDiff addresses the issues in Appendix H.2 directly. By replacing Jacobians with expected Jacobians, SmoothDiff replaces indicator functions g_p^n by smoothed, continuously differentiable functions $\widehat{g^n}_p = \left[\widehat{\boldsymbol{J}_{f^n}}\right]_p$. As a product of continuously differentiable functions, the resulting product $\hat{G}_p(\boldsymbol{x})$ is also continuously differentiable, avoiding the issue of path sensitivity (15) that characterize the SGP. Looking at sampling estimates more specifically (see Section 2.2), SmoothDiff activates all paths p in which g_p^n are non-zero in at least one sample. However, by ignoring covariances, it also

activates paths whose neurons are never jointly active in any individual sample and underestimates paths which are frequently active.

H.4 General non-linearities

While Appendix H.2 is limited to deep ReLU networks, for which G_p is an indicator function, the path sensitivity (15) also applies to general non-linearities. If a model contains non-linear functions with large second-order derivatives, the resulting terms $\partial G_p / \partial x_j$ can also become very large, directly increasing the sensitivity of path contributions and shattering the gradient. For a theoretical analysis of the influence of the curvature of f_c on gradient shattering, refer to [68].

I Example VEJP implementations

Algorithms 1 and 2 demonstrate the VEJP implementations for ReLUs and max pooling layers from Section 2.2. After replacing all `ReLU` activation functions in a model with `ReLU_SmoothDiff`, as well as replacing all `MaxPool` layers with corresponding `MaxPool_SmoothDiff`, n `forward` passes of the model are computed with samples $\hat{\boldsymbol{x}} \sim \mathcal{N}_g$, incrementing sample and activation counters. SmoothDiff can then be computed in a single `backward` pass through the model evaluating VEJPs $\boldsymbol{v}^T \hat{\boldsymbol{J}}$ (in the pseudocode, $\boldsymbol{v}^T$ is named `v` and $\hat{\boldsymbol{J}}$ is named `J`). For ReLUs, this computation corresponds to (9).

```
class ReLU_SmoothDiff
    ReLU
    sample_count = 0
    activation_count = 0
    function forward(input)
        sample_count += 1
        activation_count += input > 0
        return ReLU(input)
    function backward(v)
        J = activation_count / sample_count
        return v * J
```

Algorithm 1: SmoothDiff VEJP for ReLUs.

```
class MaxPool_SmoothDiff
    MaxPool
    sample_count = 0
    activation_count = 0
    function forward(input)
        sample_count += 1
        activation_count += is_max(input)
        return MaxPool(input)
    function backward(v)
        J = activation_count / sample_count
        return v * J
```

Algorithm 2: SmoothDiff VEJP for max pooling layers.

J Rank

For two matrices $\boldsymbol{A} \in \mathbb{R}^{m \times k}$, $\boldsymbol{B} \in \mathbb{R}^{k \times n}$, Sylvester's rank inequality tells us that

$$\operatorname{rank}(\boldsymbol{A}) + \operatorname{rank}(\boldsymbol{B}) - k \leq \operatorname{rank}(\boldsymbol{AB}) \leq \min(\operatorname{rank}(\boldsymbol{A}), \operatorname{rank}(\boldsymbol{B})) \,. \tag{16}$$

Using the factorization in (3), the rank of neural network Jacobians $\boldsymbol{J}_f$ is therefore upper-bounded by the lowest-rank Jacobian $\boldsymbol{J}_{f^n}$:

$$\operatorname{rank}\big(\boldsymbol{J}_f\big) = \operatorname{rank}\left(\prod_n \boldsymbol{J}_{f^n}\right) \leq \min_n \operatorname{rank}\big(\boldsymbol{J}_{f^n}\big) \,. \tag{17}$$

In practice, the lowest-rank Jacobians of deep ReLU networks are those of element-wise ReLU activation functions (since weight matrices tend to have full rank). As discussed in Section 2.2, ReLU Jacobians are diagonal matrices with entries $J_{\sigma_{i,i}}(\boldsymbol{a}) = \mathbb{1}_{x \geq 0}(a_i)$. Since the rank of a diagonal matrix corresponds to the number of non-zero entries, it is determined by the number of strictly positive activations a_i. SmoothDiff's expected Jacobian entries (8) are non-zero if a single sample leads to a positive activation a_i (as discussed in Section 2.2 and Appendix H), directly increasing the rank of the Jacobian computed in (5).

K Sample heatmaps

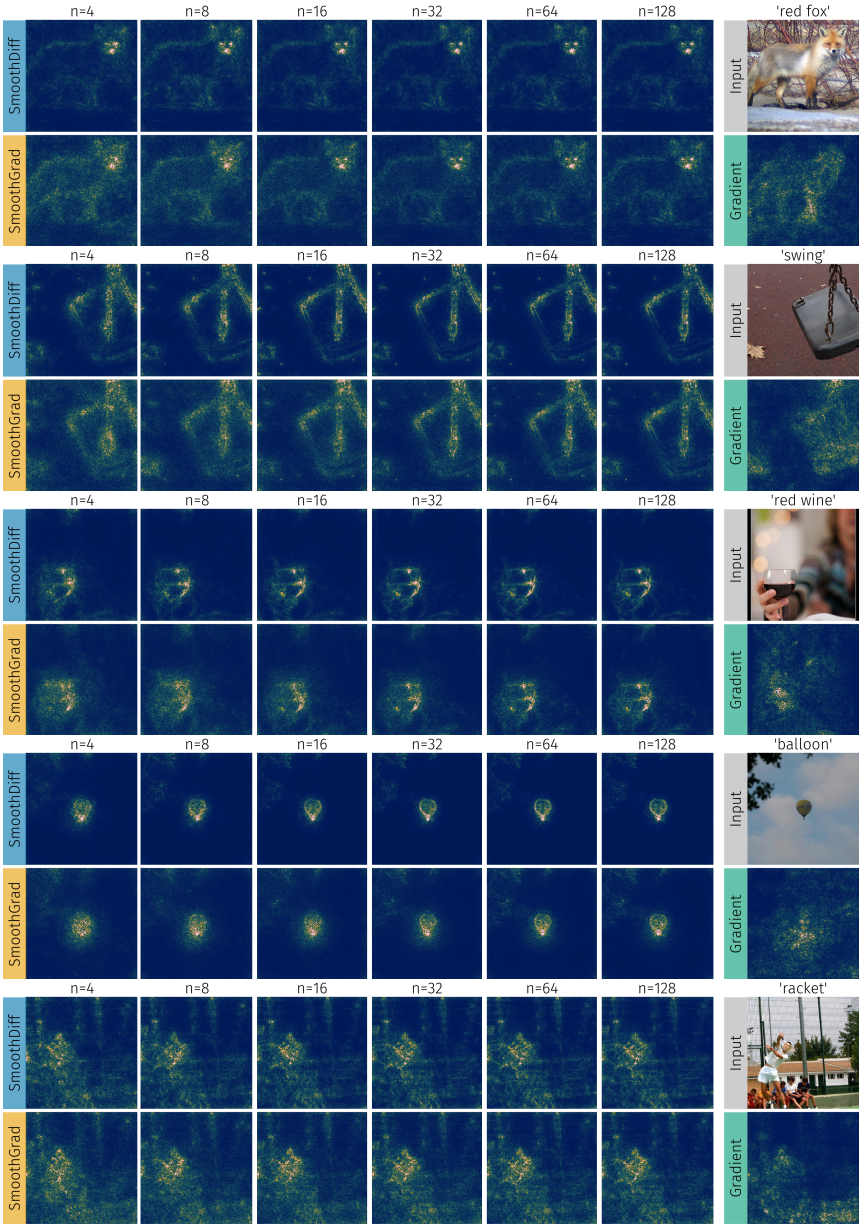

Figure 15: Qualitative comparison of SmoothDiff and SmoothGrad heatmap convergence over increasing sample sizes. Heatmaps are computed on VGG-19 using $\sigma = 0.5$.

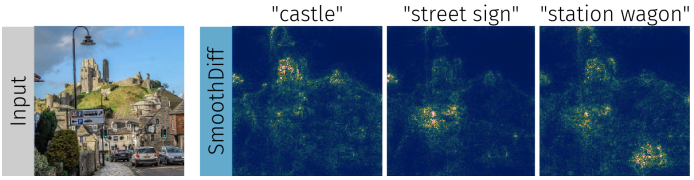

Figure 16: SmoothDiff multi-class explanations for the target classes “castle”, “street sign”, and “station wagon”. While gradient-based methods can’t distinguish between positive and negative contributions on the pixel level, explanations change significantly between classes and their highest values visibly lie within the object of the selected class. Heatmaps are computed on VGG-19 with $n = 10$ and $\sigma = 0.5$.

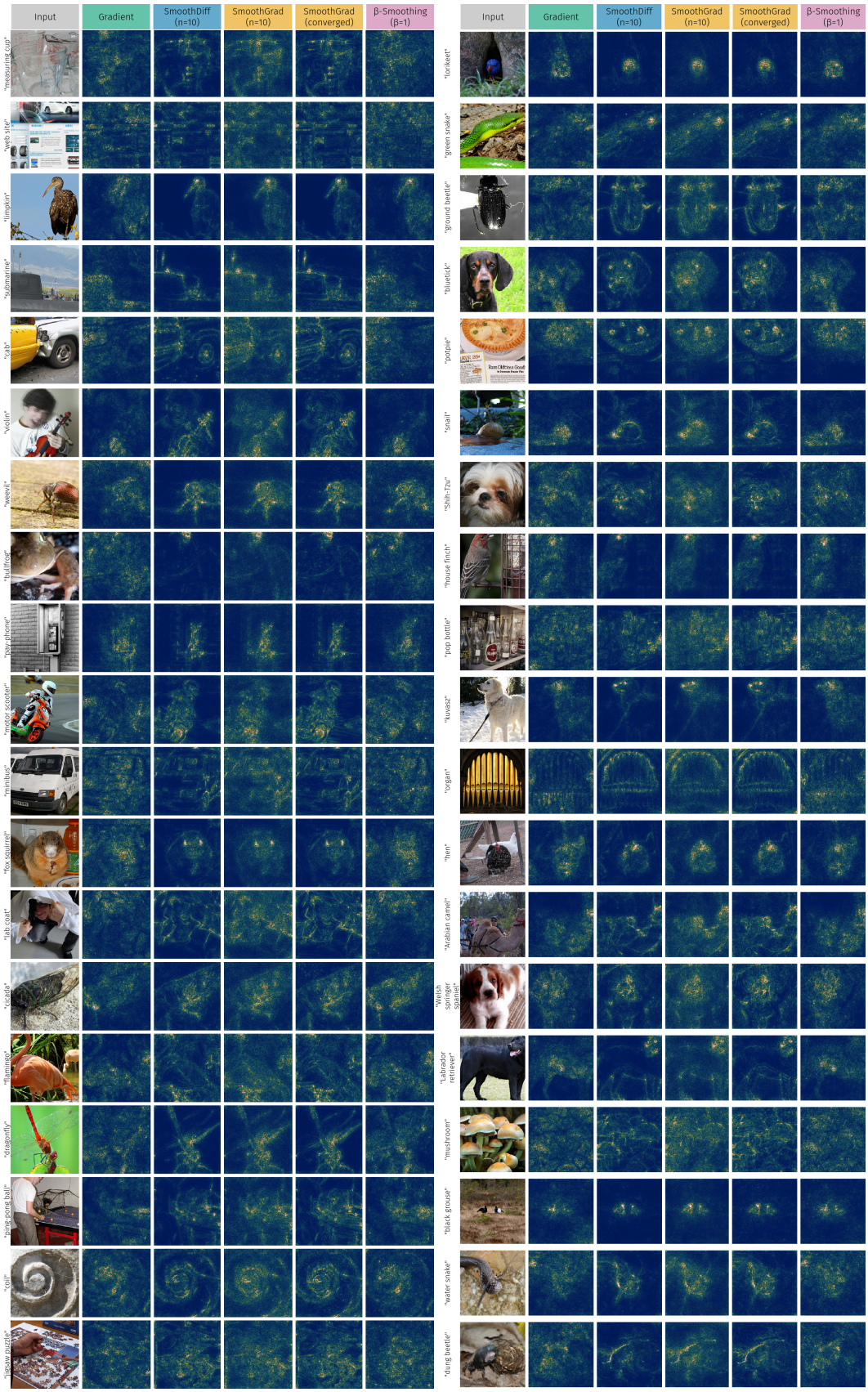

Figure 17: Comparison of gradient-based explanations on VGG-19. All SmoothDiff and SmoothGrad explanations use $\sigma = 0.5$.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [YES]

   Justification:

   - The superior convergence of SmoothDiff over SmoothGrad is demonstrated in experiments in Section 3.1 and Section 3.2, as well as qualitatively in Section 3.3.
   - The claimed performance increase of more than two orders of magnitude is derived in Section 2.4.
   - The ease of implementation is demonstrated in Section 2.2 and Appendix I
   - We demonstrate that SmoothDiff directly addresses the shattered gradient problem in Appendix H.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [YES]

   Justification:

   - Limitations on ViT are documented in Section 2.5
   - SmoothDiff efficiently approximates Gaussian convolutions by ignoring cross-covariances between Jacobians in Equation (5). This limitation is repeatedly discussed in Section 2, Section 2.1, Section 2.4 and Section 3. Neglected cross-covariances are discussed in Appendix E.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution

is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.

- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [YES]

   Justification:

   - We provide a full derivation of SmoothDiff starting from the chain rule. In Equation (5), the assumption of negligible cross-covariances is made. This assumption is repeatedly discussed in Section 2 (including Section 2.1, Section 2.4) and Section 3.
   - The paper contains no proofs or theorems.
   - Formulas are numbered and cross-referenced.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [YES]

   Justification:

   - Experiments are run on the widely available ImageNet dataset using the widely available, pre-trained VGG-19 model.
   - We provide SmoothDiff reference implementations in PyTorch and Julia.
   - The method is simple to reproduce in other code bases, as it requires very little code. Pseudocode is provided in Appendix I.
   - We provide the entire code used to run experiments, evaluations and visualizations, including virtual environments.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [YES]

   Justification:

   - We provide reference implementations of our method in PyTorch and Julia
   - We provide all scripts used to run experiments, evaluations and visualizations, including virtual environments.
   - No specialized hardware is required
   - No non-open-sourced or non-free libraries or languages are required

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).

- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [YES]

   Justification:

   - For all methods we evaluate, the full set of hyperparameters is specified (see e.g column "Parameters" in Table 1)
   - We provide all scripts for all experiments.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [NA]

   Justification:

   - The paper does not contain error bars, as the approximated Gaussian convolution has no analytic solution and is computationally intractable. However, we demonstrate that we outperform previous approximations.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [YES]

   Justification:

   - Details on compute resources used are provided in Section 3, including CPU and GPU, as well as memory.
   - The run-times of benchmarked methods are reported in Section 3.2.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`

   Answer: [YES]

   Justification:

   - We have read the NeurIPS Code of Ethics and our research fully conforms with it.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification:

- We provide a computationally more efficient alternative to the SmoothGrad method, estimating Gaussian convolutions of gradients, potentially reducing the energy consumption of such computations.
- We see no potential malicious or unintended uses.
- We see no harms that could arise from the method.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [YES]

    Justification:

    - We release no model
    - We release no dataset

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [YES]

Justification:

- We cite the VGG model architecture and ImageNet dataset in Section 3, the ImagenNet-S dataset in Appendix F
- We cite used software libraries in Section 3

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `https://paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [NA]

Justification:

- The paper does not introduce or release new assets

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification:

- The paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification:

- The paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [NA]

Justification:

- LLMs were not involved in the research beyond writing and editing assistance and as a surface level search engine to look up technical concepts.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (https://neurips.cc/Conferences/2025/LLM) for what should or should not be described.