There and Back Again: Revisiting Backpropagation Saliency Methods

Sylvestre-Alvise Rebuffi * Ruth Fong* Xu Ji* Andrea Vedaldi
Visual Geometry Group, University of Oxford
{srebuffi, ruthfong, xuji, vedaldi}@robots.ox.ac.uk

Abstract

Saliency methods seek to explain the predictions of a model by producing an importance map across each input sample. A popular class of such methods is based on backpropagating a signal and analyzing the resulting gradient. Despite much research on such methods, relatively little work has been done to clarify the differences between such methods as well as the desiderata of these techniques. Thus, there is a need for rigorously understanding the relationships between different methods as well as their failure modes. In this work, we conduct a thorough analysis of backpropagation-based saliency methods and propose a single framework under which several such methods can be unified. As a result of our study, we make three additional contributions. First, we use our framework to propose NormGrad, a novel saliency method based on the spatial contribution of gradients of convolutional weights. Second, we combine saliency maps at different layers to test the ability of saliency methods to extract complementary information at different network levels (e.g. trading off spatial resolution and distinctiveness) and we explain why some methods fail at specific layers (e.g., Grad-CAM anywhere besides the last convolutional layer). Third, we introduce a class-sensitivity metric and a meta-learning inspired paradigm applicable to any saliency method for improving sensitivity to the output class being explained.

1. Introduction

The adoption of deep learning methods by high-risk applications, such as healthcare and automated driving, gives rise to a need for tools that help machine learning practitioners understand model behavior. Given the highly-parameterized, opaque nature of deep neural networks, developing such methods is non-trivial, and there are many possible approaches. In the basic case, even the predictions of the model itself, either unaltered or after being distilled into a simpler function [11, 3], can be used to shed light on its behaviour.

Saliency is the specific branch of interpretability concerned with determining not what the behaviour of a model is for whole input samples, but which parts of samples contribute the most to that behaviour. Thus by definition, determining saliency - or attribution - necessarily involves reversing the model’s inference process in some manner [22]. Propagating a signal from the output layer of a neural network model back to the input layer is one way of explicitly achieving this.

The number of diverse works based on using signal backpropagation for interpretability in computer vision [39, 29, 4, 40] is testimony to the power of this simple principle. Typically, these techniques produce a heatmap for any given input image that ranks its pixels according to some metric of importance for the model’s decision. Inspired by the work of [1], we propose to delve deeper into such methods by discussing some of the similarities, differences and potential improvements that can be illustrated.

We begin with presenting a framework that unifies several backpropagation-based saliency methods by dividing the process of generating a saliency map into two phases: extraction of the contribution of the gradient of network parameters at each spatial location in a particular network layer, and aggregation of such spatial information into a 2D heatmap. GradCAM [29], linear approximation [18] and gradient [31] can all be cast as such processes. Noting that no appropriate technique has yet been proposed for properly aggregating contributions from convolutional layers, we introduce NormGrad, which uses the Frobenius norm for aggregation. We introduce identity layers to allow for the generation of saliency heatmaps at all spatially-grounded layers in the network (i.e. even after ReLU), since NormGrad computes saliency given a parameterised network layer.

We conduct a thorough analysis of backpropagation-based saliency methods in general, with evaluation based on utilising saliency heatmaps for weak localisation. Notably, we conduct an investigation into simple techniques for combining saliency maps taken from different network layers - in contrast to the popular practice of computing maps at the input layer [31] - and find that using a weighted aver-
age of maps from all layers consistently improves performance for several saliency methods, compared to taking the single best layer. However, not all layers are equally important, as we discover that models optimised on datasets such as ImageNet [28] and PASCAL VOC [7] learn features that become increasingly class insensitive closer towards the input layer. This provides an explanation for why Grad-CAM [29] produces unmeaningful saliency heatmaps at certain layers earlier than the last convolutional layer, as the sensitivity of the gradient to class across spatial locations is eliminated by Grad-CAM’s spatial gradient averaging, meaning such layers are devoid of class sensitive signals from which to form saliency heatmaps.

Finally, building off [22, 1], we introduce a novel metric for quantifying the class sensitivity of a saliency method, and present a meta-learning inspired paradigm that increases the class sensitivity of any method by adding an inner SGD step into the computation of the saliency heatmap.

2. Related work

Saliency methods. Our work focuses on backpropagation-based saliency methods; these techniques are computationally efficient as they only require one forward and backward pass through a network. One of the earliest methods was [31] which visualised the gradient at the input with respect to an output class being explained. Several authors have since proposed adaptations in order to improve the heatmap’s visual quality. These include modifying the ReLU derivatives (Deconvnet [39], guided backprop [34]) and averaging over randomly perturbed inputs (SmoothGrad [33]) to produce masks with reduced noise. Several works have explored visualizing saliency at intermediate layers by combining information from activations and gradients, notably CAM [41], Grad-CAM [29], and linear approximation (a.k.a. gradient × input) [18]. Conservation-preserving methods (Excitation Backprop [40], LRP [4], and DeepLIFT [30]) modify the backward functions of network layers in order to “preserve” an attribution signal such that it sums to one at any point in the network. Reference-based methods average over attributions from multiple interpolations [35] between the input and a non-informative (e.g. black) reference input or use a single reference input with which to compare a backpropagated attribution signal [30].

Although we focus on backpropagation-based methods, another class of methods studies the effects that perturbations on the input induce on the output. [39] and [26] generate saliency maps by weighting input occlusion patterns by the induced changes in model output. [10, 9, 15] learn for saliency maps that maximally impact the model, and [5] trains a model to predict effective maps. LIME [27] learns linear weights that correspond to the effect of including or excluding (via perturbation) different image patches in an image. Perturbation-based approaches have also been used to perform object localisation [32, 37, 36].

Assessing and unifying saliency methods. A few works have studied if saliency methods have certain desired sensitivities (e.g., to specific model weights [1] or the output class being explained [22]) and if they are invariant to unmeaningful factors (e.g., constant shift in input intensity [17]). [22] showed that gradient [31], deconvnet [39], and guided backprop [34] tend to produce class insensitive heatmaps. [1] introduced sanity check metrics that measure how sensitive a saliency method is to model weights by reporting the correlation between a saliency map on a trained model vs. a partially randomized model.

Other works quantify the utility of saliency maps for weak localization [40, 10] and for impacting model predictions. [40] introduced Pointing Game, which measures the correlation between the maximal point extracted from a saliency map with pixel-level semantic labels. [10] extracts bounding boxes from saliency maps and measures their agreement with ground truth bounding boxes. [38] evaluates attribution methods using relative feature importance. [24] proposes a dataset designed for measuring visual explanation accuracy. [4, 26, 15] present variants of a perturbation-based evaluation metric that measures the impact of perturbing (or unperturbing for [15]) image patches in order of importance as given by a saliency map. However, these perturbed images are outside the training domain; [13] mitigates this by measuring the performance of classifiers re-trained on perturbed images (i.e., with 20% of pixels perturbed).

To our knowledge, the only work that has been done to unify saliency methods focuses primarily on “invasive” techniques that change backpropagation rules. The α-LRP variant [4] and Excitation Backprop [40] share the backpropagation rule, and DeepLIFT [30] is equivalent to LRP when 0 is used as the reference activation throughout a network. [20] unifies a few methods (e.g., LIME [27], LRP [4], DeepLIFT [30]) under the framework of additive feature attribution.

3. Method

Preliminaries. Consider a training set D of pairs (x, y) where \( x \in \mathbb{R}^{3 \times H \times W} \) are (color) images and \( y \in \{1, \ldots, C\} \) their labels. Furthermore, let \( y = \Phi_\theta(x) \) be the output of a neural network model whose parameters \( \theta \) are optimized using the cross-entropy loss \( \ell \) to predict labels from images.

3.1. Extract & Aggregate framework

In most methods, the saliency map is obtained as a function of the network activations, computed in a forward pass,
and information propagated from the output of the network back to its input using the backpropagation algorithm. While some methods modify backpropagation in some way, here we are interested in those, such as gradient, linear approximation, and all variants of our proposed NormGrad saliency method, that do not.

In order to explain these “non-invasive” methods, we suggest that their saliency maps can be interpreted as a measure of how much the corresponding pixels contribute to changing the model parameters during a training step. We then propose a principled two-phase framework capturing this idea. In the extraction phase, a method isolates the contribution to the gradient from each spatial location. We use the fact that the gradient of spatially shared weights can be written as the sum over spatial locations of a function of the activation gradients and input features. In the aggregation phase, each spatial summand is converted into a scalar using an aggregation function, thus resulting in a saliency map.

Algorithm 1 Extract & Aggregate

1. **Extract.** Compute spatial contributions to the summation of the gradient of the weights.
   - Choose a layer whose parameters are shared spatially (layers from table 1).
   - Alternately, insert an identity layer (section 3.1.2) at the targeted location.

2. **Aggregate.** Transform these spatial contributions into saliency maps using an aggregation function.
   - Norm: NormGrad (ours)
   - Voting/Summing: linear approximation [18]
   - Filtering: GradCAM [29], selective NormGrad (ours)
   - Max: Gradient backpropagation [31]

### 3.1.1 Phase 1: Extract

We first choose a target layer in the network at which we plan to compute a saliency map. Assuming that the network is a simple chain\(^1\), we can write \( L = \ell \circ \Phi = h \circ k_w \circ q \), where \( k_w \) is the target layer parameterised by \( w \), \( h \) is the composition of all layers that follow it (including loss \( \ell \)), and \( q \) is the composition of layers that precede it. Then, \( x^{in} = q(x) \in \mathbb{R}^{K' \times H' \times W'} \) denotes the input to the target layer, and its output is given by \( x^{out} = k_w(x^{in}) \in \mathbb{R}^{K' \times H' \times W'} \).

\(^1\)Other topologies are treated in the same manner, but the notation is more complex.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Spatial contribution</th>
<th>Size at each location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bias</td>
<td>( g_u^{out} )</td>
<td>vector: ( K' )</td>
</tr>
<tr>
<td>Scaling</td>
<td>( g_u^{out} \circ x_u^{in} )</td>
<td>vector: ( K' )</td>
</tr>
<tr>
<td>Conv ( N \times N )</td>
<td>( g_u^{out} \circ x_u^{in,N \times N} )</td>
<td>matrix: ( K' \times N^2 K )</td>
</tr>
</tbody>
</table>

Table 1. Formulae and sizes of the spatial contributions to the gradient of the weights for layers with spatially shared parameters. \( \odot \) denotes the elementwise product and \( x_u^{in,N \times N} \) is the \( N^2 K \) vector obtained by unfolding the \( N \times N \) patch around the target location.

### Convolutional layers with general filter shapes.

For convolutions with an \( N \times N \) kernel size, we can re-write the convolution using the matrix form:

\[
X^{out} = WX^{in}_{N \times N}
\]

where \( X^{out} \in \mathbb{R}^{K' \times HW} \) and \( W \in \mathbb{R}^{K' \times N^2 K} \) are the output and filter tensors reshaped as matrices and \( X^{in}_{N \times N} \in \mathbb{R}^{N^2 K \times HW} \) is a matrix whose column \( x_u^{in,N \times N} \in \mathbb{R}^{N^2 K} \) contains the unfolded patches at location \( u \) of the input tensor to which filters are applied.\(^2\) Then, the gradient w.r.t. the filter weights \( W \) is given by

\[
\frac{dL}{dW} = \sum_{u \in \Omega} \frac{dL}{dW} \left( g_u^{out}, W x_u^{in,N \times N} \right) = \sum_{u \in \Omega} g_u^{out} x_u^{in,N \times N}^\top,
\]

where \( g_u^{out} = dh/dx_u^{out} \) is the gradient of the “head” of the network. Thus, for the convolutional layer case, the spatial summand is an outer product of two vectors; thus, the spatial contribution at each location to the gradient of the weights is a matrix of size \( K' \times N^2 K \).

### Other layer types.

Besides convolutional layers, bias and scaling layers also share their parameters spatially. In modern architectures, these are typically used in batch normalization layers [14]. We denote \( b, \alpha \in \mathbb{R}^K \) as the parameters for the bias and scaling layers respectively. They are defined respectively as follows:

\[
x^{out}_{ku} = x^{in}_{ku} + b_k, \quad x^{out}_{ku} = \alpha_k x^{in}_{ku}.
\]

Then, the gradients w.r.t. parameters are given by

\[
\frac{dL}{db} = \sum_{u \in \Omega} g_u^{out}, \quad \frac{dL}{d\alpha} = \sum_{u \in \Omega} g_u^{out} \odot x_u^{in}.
\]

where \( \odot \) is the elementwise product. For these two types of layer, the spatial summand is a vector of size \( K \), the number of channels. Table 1 summarizes the spatial contributions to the gradients for the different layers.

\(^2\)This operation is called `im2row` in MATLAB or `unfold` in PyTorch.
3.1.2 Virtual identity layer

So far, we have only extracted spatial gradient contributions for layers that share parameters spatially. We will now extend our summand extraction technique to any location within a CNN by allowing the insertion of a virtual identity layer at the target location. This layer is a conceptual construction that we introduce to derive in a rigorous and unified manner the equations employed by various methods to compute saliency.

We are motivated by the following question: if we were to add an identity operator at a target location, how should this operator’s parameters be changed? A virtual identity layer is a layer which shares its parameters spatially and is set to the identity. Hence, it could be any of the layer from table 1, like a bias or a scaling layer; then, \( b_k = 0 \) or \( c_k = 1 \) respectively for \( k \in \{1, \ldots, K\} \). It could also be a \( N \times N \) convolutional layer with filter bank \( W \in \mathbb{R}^{K \times N^2 K} \) such that \( w_{k,k'} \in \delta_{k,k'} \delta_{i,o} \delta_{j,0} \), where \( \delta_{a,b} \) is the Kronecker delta function\(^3\), for \((i,j) \in \{-\frac{N-1}{2}, \ldots, \frac{N-1}{2}\}\).

Because of the nature of the identity, this layer does not change the information propagated in either the forward or backward direction. Conceptually, it is attached to the part of the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wish to inspect as shown in fig. 1. Because of the nature of the identity, this layer does not change the information propagated in either the forward or backward direction. Conceptually, it is attached to the part of the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in fig. 1. This layer is never “physically” added to the model (i.e., the model that one wishes to inspect as shown in Fig. 1. Virtual identity. Here, we visualize inserting an identity after a specific layer in the network for saliency computation purposes. The gradient coming from later stages of the network that is the gradient at output of the network layer, is now also the gradient at the output of the identity, \( g^{out} \).

3Kronecker delta function: \( \delta_{a,b} = 1 \) if \( a = b \); otherwise, \( \delta_{a,b} = 0 \).

3.1.3 Phase 2: Aggregate

Following the extraction phase (section 3.1.1), we have the local contribution at each spatial location to the gradients of either an existing layer or a virtual identity layer. Each spatial location is associated with a vector or matrix (table 1). In this section, we describe different aggregation functions to map these vectors or matrices to a single scalar per spatial position. We drop the spatial location \( u \) in the notations.

Understanding existing saliency methods. One possible aggregation function is the sum of the elements in a vector. When combined with a virtual scaling identity layer (section 3.1.2), we obtain the linear approximation method [18]: \( \sum_k g_k^{out} x_k^{out} \). The contributions from the scaling identity encode the result of channels changing (after the gradient is applied) at each location; thus, the sum aggregation function acts as a voting mechanism. The resulting saliency map highlights the locations that would be most impacted if following through with the channel updates.

Aggregation functions can also be combined with element-wise filtering functions (e.g., the absolute value function). Another aggregation function takes the maximum absolute value of the vector: \( \maxabs(x) = \max \{ |x_k| \} \). If we combine a virtual bias identity layer in phase 1, which gives \( g^{out} \) as the spatial contribution, with the \( \maxabs \) function for aggregation, we obtain at each spatial location the gradient [31] method: \( \max_k |g_k^{out}| \).

As for CAM [41] and Grad-CAM [29], we cannot directly use the spatial contributions extracted at each location because they spatially average \( g^{out} \). However, for architectures that use global average pooling followed after their convolutional layers (e.g., ResNet architectures), \( g_k^{out} = g^{out} \). Then, CAM and Grad-CAM and CAM can be viewed as combining a virtual scaling identity layer from phase 1 with summing and positive filtering (i.e., \( \text{filter}_+(x) = (x)_+ \)) functions for aggregation.

NormGrad. The sum and \( \max \) functions have clear interpretations when using bias or scaling identity layers; however, they cannot be easily transported to convolutional identity layers as interactions between input channels can vary depending on the output channel and are represented as a matrix, not as a vector as are the case for scaling and bias layers. Thus, we would like to have an aggregation function that could be used to aggregate any type of spatial contribution, regardless of its shape.

Using the norm function satisfies this criterion, for example the \( L^2 \) norm when dealing with vectors and the Frobenius norm for matrices. We note that the matrices obtained at each location for convolutional layers are the outer

\(^4\)Because CAM global average pooling + one fully connected layer, \( g^{out} \) is equal to the fully connected weights.
products of two vectors. For such matrices, the Frobenius norm is equal to the product of the norms of the two vectors. For example, for an existing $1 \times 1$ convolutional layer, we consider the saliency map given by $\|g^\text{out}\||x^\text{in}\|$. We call this class of saliency methods derived by using norm as an aggregation function, “NormGrad”.

Saliency maps from the NormGrad outlined above are not as class selective as other methods because they highlight the spatial locations that contribute the most to gradient of the weights, both positively and negatively. One way to introduce class selectivity is to use positive filtering before applying the norm. If we apply norm and positive filtering aggregation to a scaling identity layer, the resulting saliency map is given by $\|g^\text{out} \circ x^\text{out}\|$. Throughout the rest of the paper, we call this variant “selective NormGrad”.

### 4.1. Justifying the virtual identity trick.

In order to justify our use of the virtual identity trick, we compare NormGrad saliency maps generated at $1 \times 1$ and $3 \times 3$ convolutional layers both with and without the virtual identity trick (4th and 5th rows respectively in table 2) for VGG16 and ResNet50. We first computed the correlations between saliency maps generating with and without the virtual identity trick. We found that the mean correlation across $N = 50k$ ImageNet validation images was over 95% across all layers we tested. We also evaluated their performance on the Pointing Game [40] and found that the mean absolute difference in pointing game accuracy was $0.53\% \pm 0.62\%$ across all layers we tested (see supp. for more details and full results). This empirically demonstrates that using the virtual identity trick closely approximates the behavior of calculating the spatial contributions for the original convolutional layers.

### 4.2. Combining saliency maps across layers.

Training linear classifiers on top of intermediate representations is a well-known method for evaluating the learned features of a network at different layers [2]. This suggests that saliency maps, too, may have varying levels of meaningfulness at different layers.

We explore this question by imposing several weighting methods for combining the layer-wise saliency maps of ResNet50 and VGG16 and measuring the resulting performance on PASCAL VOC Pointing Game on both the “difficult” and “all” image sets. To determine the weight $\gamma_j$ for a given layer $j$ out of $J$ layers in a network we use:

1. **Feature spread.** Given the set of feature activations at layer $j$, $x^i$ for $i \in M$ input images sampled uniformly across classes, compute the spatial mean $\bar{x} = \sum_{u \in \Omega} x^i_u$. $\gamma_j = \frac{1}{\bar{x}} \sum_{i=1}^M |\bar{x}^i - \bar{x}|$ where $\bar{x}^i = \frac{1}{\bar{x}} \sum_{u \in \Omega} x^i_u$.

2. **Classification accuracy.** Given the set of feature activations at layer $j$, $x^i$ for $i \in M$ input images sampled uniformly across classes, train a linear layer $\Psi$ using image labels $y^i$. $\gamma_j = \frac{1}{\bar{y}} \sum_{i=1}^M \delta_{\Psi(x^i),y^i}$ where $\delta_{\Psi(x^i),y^i} = 1$ if $\Psi(x^i) = y^i$, 0 otherwise.

3. **Linear interpolation.** $\gamma_j = \frac{j}{J}$.

4. **Uniform.** $\gamma_j = \frac{1}{J}$.

To obtain a combined saliency map $M$ from maps $m_j$ from each layer $j$, the weights are normalised and applied either additively, $M = \sum_{j=0}^J \gamma_j \cdot m_j$, or with a product, $M = \prod_{j=0}^J m_j$ (see fig. 2 for visualization).

Table 3 shows that weighted saliency maps produce the best overall performance in all four key test cases, which is surprising as there were only four weight schemes tested.

### Table 2: Combining layers and aggregation functions for saliency. $g^\text{out}$ and $x^\text{out}$ are tensor slices for channel $f$ and contain only spatial information. IDL denotes an identity layer. F + N is positive filtering + norm.

<table>
<thead>
<tr>
<th>Phase 1</th>
<th>Phase 2</th>
<th>Saliency map</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bias IDL</td>
<td>Max</td>
<td>$\max_k g^\text{out}_k$</td>
</tr>
<tr>
<td>Scaling IDL</td>
<td>Sum</td>
<td>$\sum_k (g^\text{out}_k \circ x^\text{out}<em>k)</em>+$</td>
</tr>
<tr>
<td>Scaling IDL</td>
<td>F + N</td>
<td>$|g^\text{out}|</td>
</tr>
<tr>
<td>Conv $1 \times 1$ IDL</td>
<td>Norm</td>
<td>$|g^\text{out}|</td>
</tr>
<tr>
<td>Real Conv $3 \times 3$</td>
<td>Norm</td>
<td>$|g^\text{out}|</td>
</tr>
</tbody>
</table>

Additional experiments such as image captioning visualizations are included in the appendix.

### Experimental set-up.

Unless otherwise stated, saliency maps are generated on images from either the PASCAL VOC [7] 2007 test set or the ImageNet [6] 2012 validation set for either VGG16 [31] or ResNet50 [12]. For PASCAL VOC, we use a model pre-trained on ImageNet with fine-tuned fully connected layers on PASCAL VOC. We use [9]’s TorchRay interpretability package to generate saliency methods for all other saliency methods besides our NormGrad and meta-saliency methods as well as to evaluate saliency maps on [40]’s Pointing Game (see [40] for more details). For all correlation analysis, we compute the Spearman’s correlation coefficient [23] between saliency maps that are upscaled to the input resolution: $224 \times 224$. 
Figure 2. Saliency maps at different network depths and as a weighted combination. Linear approximation vs. selective NormGrad saliency maps of VGG16 on VOC2007. The first 5 images of each row correspond to different depths whereas the last one is a weighted product combination (using classification accuracy weights) of the first saliency maps. We observe that the weighted version produces more fined grained maps for both methods.

Figure 3. Select Pointing Game results. Results for ResNet50 on VOC07 at different network depths (A: all images; D: difficult subset). Grad-CAM performs worse at every layer except the last convolutional layer and lower than pointing at the center (all: 69.6%; diff: 42.4%) at most layers.

Table 3. Pointing game results on VOC07. b.s. and b.w. stand for best single layer and best weighted combination. (C)EB: (Contrastive) Excitation Backprop, GC: GradCAM, Gd(s): Gradient (sum), Gui: guided backprop, LA: linear approximation, (s)NG: (selective) NormGrad.

<table>
<thead>
<tr>
<th></th>
<th>All</th>
<th>Difficult</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ResNet50</td>
<td>VGG16</td>
</tr>
<tr>
<td></td>
<td>b.s.</td>
<td>b.w.</td>
</tr>
<tr>
<td>CEB</td>
<td>90.7</td>
<td>88.6</td>
</tr>
<tr>
<td>EB</td>
<td>84.5</td>
<td>83.1</td>
</tr>
<tr>
<td>GC</td>
<td>90.3</td>
<td>90.5</td>
</tr>
<tr>
<td>Gd</td>
<td>83.9</td>
<td>83.3</td>
</tr>
<tr>
<td>Gds</td>
<td>80.0</td>
<td>77.4</td>
</tr>
<tr>
<td>Gui</td>
<td>82.3</td>
<td>81.0</td>
</tr>
<tr>
<td>LA</td>
<td>90.2</td>
<td>91.2</td>
</tr>
<tr>
<td>NG</td>
<td>84.6</td>
<td>83.5</td>
</tr>
<tr>
<td>sNG</td>
<td>87.4</td>
<td>88.7</td>
</tr>
</tbody>
</table>

Note that the feature spread and classification accuracy metrics can both be used as indicators of class sensitivity (section 4.3). This is because if feature activations are uniform for images sampled across classes, it is not possible for them to be sensitive to - or predictive of - class, and the classification accuracy metric is an explicit quantisation of how easily features can be separated into classes. We observe from the computed weights that both metrics generally increase with layer depth (see supp.).
Grad-CAM spatially averages the backpropagated gradient before taking a product with activations, each pixel location in the heatmap receives the same gradient vector (across channels) irrespective of the image content contained within its receptive field. Thus, if the activation map used in the ensuing product is also not class selective - firing on both dogs and cats for example, fig. 4 - the saliency map cannot be. On the other hand, methods that do not spatially average gradients such as NormGrad (fig. 3) can rely on gradients that are free to vary across the heatmap with underlying class, increasing the class sensitivity of the resulting saliency map.

4.3. An explicit metric for class sensitivity

![Figure 5. Class sensitivity with and without meta-saliency. Min class saliency maps that use meta-saliency (row 2 and 4, right col) are less informative than those that don’t use meta-saliency (rows 1 and 3, right col). Class 1 is the ground truth class (fence), class 2 is the maximally predicted class (Cardigan Welsh corgi), min class is the minimally predicted class (black widow spider).](image)

[22] qualitatively shows that early backprop-based methods (e.g., gradient, deconvnet, and guided backprop) are not sensitive to the output class being explained by showing that saliency maps generated w.r.t. different output classes and gradient signals appear visually indistinguishable. Thus, similar to [1], we introduce a sanity check to measure a saliency method’s output class sensitivity. We compute the correlation between saliency maps w.r.t. to output class predicted with highest confidence (max class) and that predicted with lowest confidence (max and min class respectively) for \( N = 1000 \) ImageNet val. images (1 per class).

We would expect saliency maps w.r.t. the max class to be visually salient while those w.r.t. to the min class to be uninformative (because the min class is not in the image). Thus, we desire the correlation scores to be close to zero.

![Figure 6. Class sensitivity of saliency methods. This plot shows the correlation between VGG16 saliency maps computed w.r.t. to the maximally and minimally predicted class (closer to zero is better).](image)

Figure 6 shows results for various saliency methods. We observe that excitation backprop and guided backprop yield correlation scores close to 1 for all layers, while contrastive excitation backprop yields scores closest to 0. Furthermore, methods using sum aggregation (e.g., gradient [sum], linear approx, and Grad-CAM) have negative scores (i.e., their max-min-class saliency maps are anti-correlated). This is because sum aggregation acts as a voting mechanism; thus, these methods reflect the fact that the network has learned anti-correlated relationships between max and min classes.

4.4. Meta-saliency analysis

As a general method for improving the sensitivity of saliency heatmaps to the output class used to generate the predicted with highest confidence (max class) and that predicted with lowest confidence (max and min class respectively) for \( N = 1000 \) ImageNet val. images (1 per class).

We would expect saliency maps w.r.t. the max class to be visually salient while those w.r.t. to the min class to be uninformative (because the min class is not in the image). Thus, we desire the correlation scores to be close to zero.

![Figure 7. Meta-saliency improves class sensitivity for all saliency methods. Using meta-saliency yields weaker correlations between the saliency maps w.r.t. to the maximally and minimally predicted class (closer to zero is better).](image)
gradient, we propose to perform an inner SGD step before computing the gradients with respect to the loss. This way we can extend any saliency method to second order gradients. This is partly inspired by the inner step used in, for example, few shot learning [8] and architecture search [19].

We want to minimize:

\[ L(\theta, x) = \ell(\theta - \epsilon \nabla_\theta \ell(\theta, x), x). \]  

(4)

We take \( \epsilon \ll 1 \) to use a Taylor expansion of this loss at \( \theta \) and we now have the resulting approximated loss:

\[ L(\theta, x) \approx \ell(\theta, x) - \epsilon \| \nabla_\theta \ell(\theta, x) \|^2. \]  

(5)

As done in the previous section, we can now take the gradient of the loss with respect to the parameters \( \theta \):

\[ \nabla_\theta L(\theta, x) \approx \nabla_\theta \ell(\theta, x) - 2\epsilon \nabla_\theta^2 \ell(\theta, x) \nabla_\theta \ell(\theta, x). \]  

(6)

Using a finite difference scheme of step \( h \) as in [25], we can approximate the hessian-vector product by:

\[ \nabla_\theta^2 \ell(\theta, x) \nabla_\theta \ell(\theta, x) = \frac{\nabla_\theta \ell(\theta, x) - \nabla_\theta \ell(\theta - h \nabla_\theta \ell(\theta, x))}{h} + O(h). \]

where \( \theta^- = \theta - h \nabla_\theta \ell(\theta, x) \). We chose on purpose a backward finite difference such that two terms cancel each other when taking \( h = 2\epsilon \) and we get:

\[ \nabla_\theta L(\theta, x) \approx \nabla_\theta \ell(\theta', x). \]

where \( \theta' = \theta - 2\epsilon \nabla_\theta \ell(\theta, x) \) corresponds to one step of SGD of learning rate \( 2\epsilon \). We notice that if we take \( \epsilon \to 0 \), this formula boils back down to the original gradient of the weights without meta step. We further note that this meta saliency approach only requires one more forward-backward pass compared to usual saliency backpropagation methods.

Conversely, if we would like to get an importance map that highlights the degradation of the model’s performance, we should add an inner step with gradient ascent within the loss. Hence by minimizing the resulting loss \(-\ell(\theta + \epsilon \nabla_\theta \ell(\theta, x), x)\), we get the same formula for the gradients of the weights but with \( \theta' = \theta + 2\epsilon \nabla_\theta \ell(\theta, x) \).

We hypothesize that applying meta-saliency to a saliency method should decrease correlation strength because allowing the network to update one SGD step in the direction of the min class should “destroy” the informativeness of the resulting saliency map. We use a learning rate \( \epsilon = 0.001 \) for the class sensitivity quantitative analysis. Figure 5 shows qualitatively that this appears to be the case: without meta-saliency, selective NormGrad and linear approximation yield max (class 2) and min class heatmaps that are highly positively and negatively correlated respectively. However, when meta-saliency is applied, the min class saliency map appears more random. Figure 7 shows results comparing the max-min class correlation scores with and without meta-saliency. These results demonstrate that meta-saliency decreases max-min class correlation strength for nearly all saliency methods and suggest that meta-saliency can increase the class sensitivity for any saliency method.

4.5. Model weights sensitivity

[1] shows that some saliency methods (e.g., Guided Backprop in particular) are not sensitive to model weights as they are randomized in a cascading fashion from the end to the beginning of the network. Figure 8 shows qualitatively that, by the late conv layers, saliency maps for linear approximation and selective NormGrad are effectively scrambled (top two rows). It also highlights that, because meta-saliency increases class selectivity and is allowed to take one SGD in the direction of the target class, it takes relatively longer (i.e., more network depth) to randomize a meta-saliency heatmap (bottom row and see appendix).

5. Conclusions

We introduced a principled framework based on the contribution of each spatial location to the weights’ gradient. This framework unifies several existing backpropagation-based methods and allowed us to systematically explore the space of possible saliency methods. We use it for example to formulate NormGrad, a novel saliency method. We also studied how to combine saliency maps from different layers, discovering that it can consistently improve weak localization performance and produce high resolution maps. Finally, we introduced a class-sensitivity metric and proposed meta-saliency, a novel paradigm applicable to any existing method to improve sensitivity to the target class.

6. Acknowledgments

This work is supported by Mathworks/DARPA, Open Philanthropy, EPSRC AIMS CDT and ERC 638009-IDIU.
References


[37] Yunchao Wei, Jiashi Feng, Xiaodan Liang, Ming-Ming Cheng, Yao Zhao, and Shuicheng Yan. Object region mining and distractors on the interpretation of neural networks. arXiv, 2016. 1, 2, 3, 4, 5

[38] Mengjiao Yang and Been Kim. Benchmarking attribution methods with relative feature importance. arXiv, 2019. 2


A. Pointing Game details

We use [40]'s Pointing Game for quantitative evaluation of saliency methods. Saliency maps are computed with respect to every object class present in each image. If the maximally salient point for each class lands on the ground truth annotation for that object (within a threshold of 15 pixels), then a “hit” is recorded; otherwise, a “miss” is recorded. Pointing game accuracy is computed as the mean over per-class accuracies given by the following:

\[
\frac{|\text{hits}|}{|\text{hits} + \text{misses}|}.
\]

We evaluate VGG16 and ResNet50 networks that have been trained on ImageNet and fine-tuned on PASCAL VOC and COCO. We evaluate on the PASCAL VOC 2007 test split (\(N = 4952\) images) and COCO 2014 val split (\(N \approx 50\)k). We also show performance on the difficult subsets of the data provided by [40]; these are images for which the total area of the annotations (bounding boxes for PASCAL VOC and segmentation masks for COCO) for the given object class is less than 25% of the image size and for which there is at least one other object class present. We use [9]'s TorchRay library for evaluation; see [40] for more details.

B. Virtual identity trick correlations

We computed the correlation between saliency maps generated both with and without the virtual identity trick (paper section 4.1). The high correlation shown in Table 4, as well as the minimal difference in pointing game performance (0.53% ± 0.62%) demonstrates that the identity trick closely approximates the behaviour of calculating the spatial contributions for the original convolutional layers.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Layer</th>
<th>Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet50</td>
<td>layer1.0.conv1</td>
<td>99.48 ± 0.55</td>
</tr>
<tr>
<td>ResNet50</td>
<td>layer2.0.conv1</td>
<td>99.32 ± 0.35</td>
</tr>
<tr>
<td>ResNet50</td>
<td>layer3.0.conv1</td>
<td>99.16 ± 0.42</td>
</tr>
<tr>
<td>ResNet50</td>
<td>layer4.0.conv1</td>
<td>98.35 ± 1.05</td>
</tr>
<tr>
<td>VGG16</td>
<td>features.2</td>
<td>97.77 ± 1.34</td>
</tr>
<tr>
<td>VGG16</td>
<td>features.7</td>
<td>99.15 ± 0.42</td>
</tr>
<tr>
<td>VGG16</td>
<td>features.14</td>
<td>98.99 ± 0.57</td>
</tr>
<tr>
<td>VGG16</td>
<td>features.21</td>
<td>97.79 ± 1.42</td>
</tr>
<tr>
<td>VGG16</td>
<td>features.28</td>
<td>94.33 ± 3.09</td>
</tr>
</tbody>
</table>

Table 4. Correlations between saliency maps generated with and without the virtual identity trick. For both VGG16 and ResNet50, the high values of correlation across the network (min of 94.33) justify the use of the identity trick.

C. Performance of combining saliency maps

As noted in paper section 4.2, feature spread and classification accuracy are both interpretable as measures of feature sensitivity with respect to the class of the input image. Figure 9 shows that both increase with network depth, with the exception of feature spread for VGG16, which decreases in the last two layers despite simultaneously increasing classification accuracy. We note that classification accuracy is a more reliable metric than feature spread as it directly codes for the separability of features with respect to class, whereas feature spread is susceptible to non-material differences in the absolute scale of activation values across layers. In the case of VGG16, this means the features in the last two layers differ less across images (and hence, classes) in absolute terms, but are nonetheless highly discriminative of class.

The performance gains of our weighting schemes over using the best individual layer are given in table 6. The best individual layers for computing maps are given in table 5, which is notable as in no case does the practice of computing saliency at the earliest layer produce the best performance.

D. Meta-saliency analysis

We hypothesized that meta vs. non-meta saliency maps should be less correlated for validation images than for training images. This is because the inner gradient step of meta-saliency should not be as impactful for a seen training image as for an unseen validation image.

For both NormGrad and selective NormGrad, we evaluated over time the average correlation between the impor-
E. Model weights sensitivity

In fig. 13, we show the effects of cascading randomization on linear approximation and selective NormGrad methods by using the same images as [1] ("junco", "corn" and "Irish terrier") to illustrate the model weights sensitivity.

Qualitatively, both methods - with and without meta-saliency - demonstrate the desired sensitivity to model weights as the saliency maps progressively lose focus from the object target as layer depth decreases. We also observe that using meta-saliency on top of the chosen base saliency method (shown on every other line of fig. 13) delays the degradation in saliency maps to earlier layers.

F. Image captioning

We show in this section a few examples of our proposed selective NormGrad method applied to the image captioning setting. We use a variant [21] of the original neuralTalk2 [16] with a ResNet101 as backbone network followed by the LSTM caption model. As in [29] we do a backward pass using the log probability of the generated caption as objective function. We apply selective NormGrad before the final global average pooling and at the ReLU layers just after the downsampling shortcuts of the third and fourth macro blocks. We also compare these saliency maps with the product combination map of these layers using a linear weighting scheme (see section 4.2 in the paper). We notice in fig. 14 that using a combination of layers produces a sharper saliency map than for individual layers.
Figure 10. Correlation between meta and non-meta saliency maps over time for train and val splits. For both NormGrad (top) and selective NormGrad (bottom), the correlation is lower on the validation split.

Figure 11. Mean correlation on the val split between the meta-saliency maps of selective NormGrad at epoch $t$ and the end of training. The saliency maps stabilize at the end of the training.
Figure 12. **Examples of positive and negative meta-saliency for selective NormGrad.** Negative meta-saliency (left images) corresponds to a gradient ascent inner step whereas a gradient descent step is used for the positive meta-saliency (images on the right). The center images do not use meta-saliency.
Figure 13. Model weights sensitivity on VGG16. Linear Approximation and selective NormGrad with and without meta-saliency.
Figure 14. **Image captioning explanations.** Selective NormGrad is applied at different layers or combination of layers (last column) of a ResNet101. We observe that saliency maps at individual layers highlight a big part of the images. Using a combination of layers allows a clearer focus on the important parts of the images.
Figure 15. **Visualizations for some methods of the Extract & Aggregate framework at different layers of VGG16.** The gradient backpropagation method (first row) works well at all stages except at the end of the network. Selective NormGrad (last row), NormGrad (third row) and Linear Approximation (second row) perform well across the network. Finally, we can observe that selective NormGrad and Linear Approximation are more class selective than NormGrad as the non-targeted "tiger cat" appears more in the third row.