Understanding Deep Networks via Extremal Perturbations and Smooth Masks

Ruth Fong[†*] University of Oxford
Mandela Patrick[†] University of Oxford
Andrea Vedaldi Facebook AI Research

Abstract

The problem of attribution is concerned with identifying the parts of an input that are responsible for a model's output. An important family of attribution methods is based on measuring the effect of perturbations applied to the input. In this paper, we discuss some of the shortcomings of existing approaches to perturbation analysis and address them by introducing the concept of extremal perturbations, which are theoretically grounded and interpretable. We also introduce a number of technical innovations to compute extremal perturbations, including a new area constraint and a parametric family of smooth perturbations, which allow us to remove all tunable hyper-parameters from the optimization problem. We analyze the effect of perturbations as a function of their area, demonstrating excellent sensitivity to the spatial properties of the deep neural network under stimulation. We also extend perturbation analysis to the intermediate layers of a network. This application allows us to identify the salient channels necessary for classification, which, when visualized using feature inversion, can be used to elucidate model behavior. Lastly, we introduce TorchRay[1], an interpretability library built on PyTorch.

1. Introduction

Deep networks often have excellent prediction accuracy, but the basis of their predictions is usually difficult to understand. *Attribution* aims at characterising the response of neural networks by finding which parts of the network's input are the most responsible for determining its output. Most attribution methods are based on backtracking the network's activations from the output back to the input, usually via a modification of the backpropagation algorithm [23, 31, 26, 32, 22, 3]. When applied to computer vision models, these methods result in *saliency maps* that highlight important regions in the input image.

However, most attribution methods do not start from a definition of what makes an input region important for the neural network. Instead, most saliency maps are validated *a-posteriori* by either showing that they correlate with the image content (e.g., by highlighting relevant object categories), or that they find image regions that, if perturbed, have a large effect on the network's output (see Sec. 2).

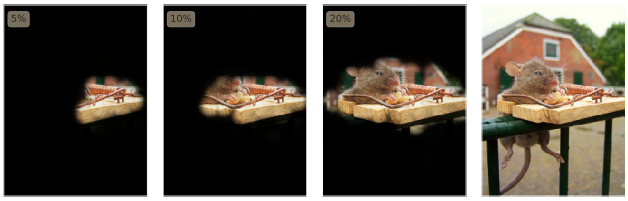

Figure 1: **Extremal perturbations** are regions of an image that, for a given area (boxed), maximally affect the activation of a certain neuron in a neural network (i.e., "mousetrap" class score). As the area of the perturbation is increased, the method reveals more of the image, in order of decreasing importance. For clarity, we black out the masked regions; in practice, the network sees blurred regions.

Some attribution methods, on the other hand, directly perform an analysis of the effect of *perturbing* the network's input on its output [31, 20, 7, 5]. This usually amounts to selectively deleting (or preserving) parts of the input and observing the effect of that change to the model's output. The advantage is that the meaning of such an analysis is clear from the outset. However, this is not as straightforward as it may seem on a first glance. First, since it is not possible to visualise *all* possible perturbations, one must find *representative* ones. Since larger perturbations will have, on average, a larger effect on the network, one is usually interested in small perturbations with a large effect (or large perturbations with a small effect). Second, Fong and Vedaldi [7] show that searching for perturbations with a large effect on the network's output usually results in *pathological* perturbations that trigger adversarial effects in the network. Characterizing instead the *typical* behavior of the model requires restricting the search to more representative perturbations via regularization terms. This results in an optimization problem that trades off maximizing the effect of the perturbation with its smoothness and size. In practice, this trade off is difficult to control numerically and somewhat obscures the meaning of the analysis.

In this paper, we make three contributions. First, instead

*Work done as a contractor at FAIR. † denotes equal contributions.
[1] https://github.com/facebookresearch/TorchRay

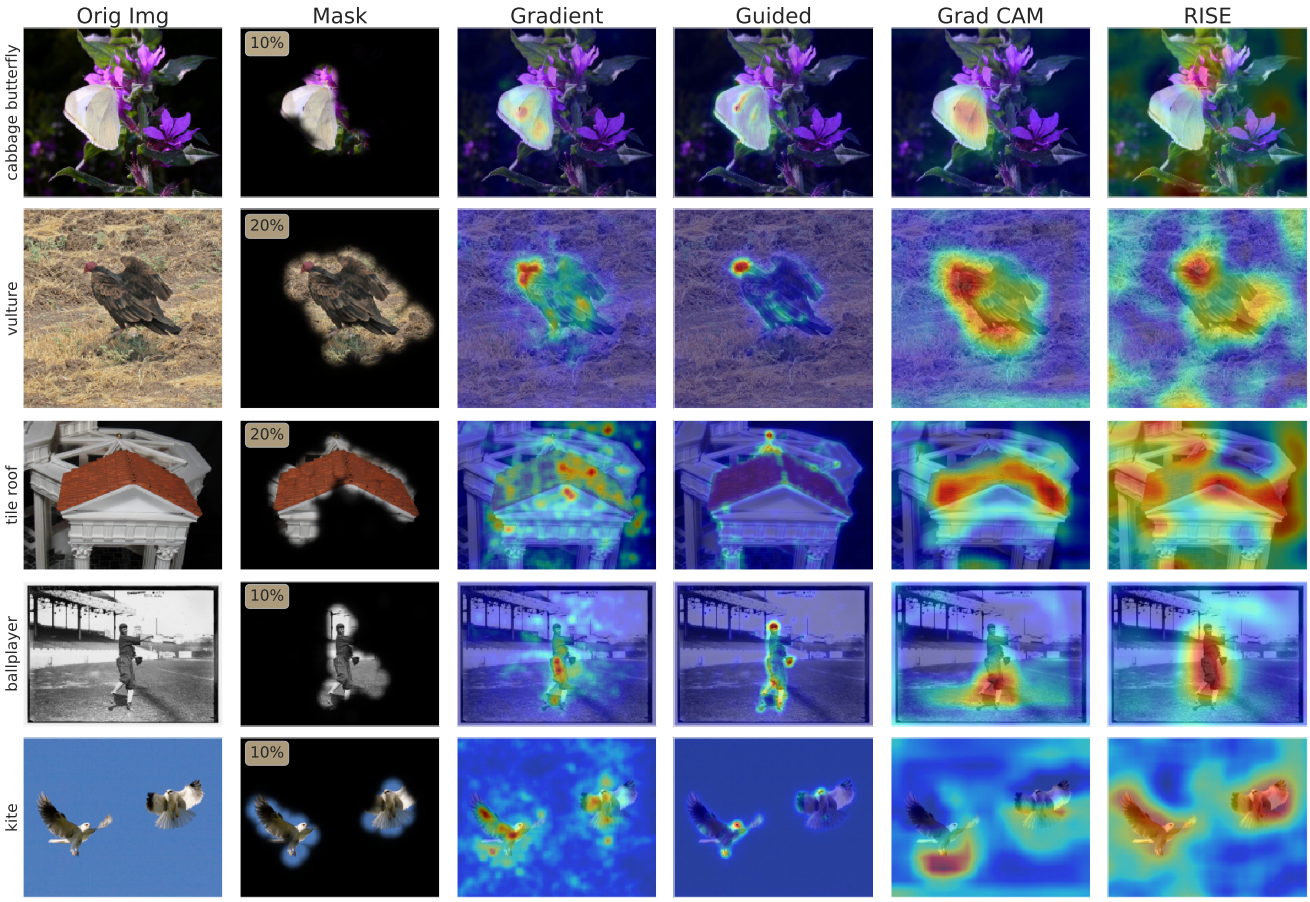

Figure 2: **Comparison with other attribution methods.** We compare our extremal perturbations (optimal area a^* in box) to several popular attribution methods: gradient [23], guided backpropagation [26], Grad-CAM [22], and RISE [20].

of mixing several effects in a single energy term to optimize as in Fong and Vedaldi [7], we introduce the concept of *extremal perturbations*. A perturbation is extremal if it has maximal effect on the network's output among all perturbations of a given, fixed area. Furthermore, the perturbations are regularised by choosing them within family with a minimum guaranteed level of smoothness. In this way, the optimisation is carried over the perturbation effect only, without having to balance several energy terms as done in [7]. Lastly, by sweeping the area parameter, we can study the perturbation's effect w.r.t. its size.

The second contribution is technical and is to provide a concrete algorithm to calculate the extremal perturbations. First, in the optimisation we must *constrain* the perturbation size to be equal to a target value. To this end, we introduce a new ranking-based *area loss* that can enforce these type of constraints in a stable and efficient manner. This loss, which we believe can be beneficial beyond our perturbation analysis, can be interpreted as a hard constraint, similar to a logarithmic barrier, differing from the soft penalty on the area in Fong and Vedaldi [7]. Second, we construct a parametric family of perturbations with a minimum guarantee amount of smoothness. For this, we use the *(smooth)-max-convolution operator* and a *perturbation pyramid*.

As a final contribution, we extend the framework of perturbation analysis to the intermediate activations of a deep neural network rather than its input. This allows us to explore how perturbations can be used beyond spatial, input-level attribution, to channel, intermediate-layer attribution. When combined with existing visualization techniques such as feature inversion [13, 19, 16, 28], we demonstrate how intermediate-layer perturbations can help us understand which channels are salient for classification.

2. Related work

Backpropagation-based methods. Many attribution techniques leverage backpropagation to track information from the network's output back to its input, or an intermediate layer. Since they are based on simple modifications of the backpropagation algorithm, they only require a single forward and backward pass through the model, and are thus efficient. [23]'s gradient method, which uses unmodified backprop, visualizes the derivative of the network's output

w.r.t. the input image. Other works (e.g., DeCovNet [31], Guided Backprop [26], and SmoothGrad [25]) reduce the noise in the gradient signal by tweaking the backprop rules of certain layers. Other methods generate visualizations by either combining gradients, network weights and/or activations at a specific layer (e.g., CAM [33] and Grad-CAM [22]) or further modify the backpropn rules to have a probabilistic or local approximation interpretation (e.g., LRP [3] and Excitation Backprop [32]).

Several papers have shown that some (but not all) backpropagation-based methods produce the same saliency map regardless of the output neuron being analysed [14], or even regardless of network parameters [2]. Thus, such methods may capture average network properties but may not be able to characterise individual outputs or intermediate activations, or in some cases the model parameters.

Perturbation-based methods. Another family of approaches perturbs the inputs to a model and observes resultant changes to the outputs. Occlusion [31] and RISE [20] occlude an image using regular or random occlusions patterns, respectively, and weigh the changes in the output by the occluding patterns. Meaningful perturbations [7] optimize a spatial perturbation mask that maximally affects a model's output. Real-time saliency [5] builds on [7] and learns to predict such a perturbation mask with a second neural network. Other works have leveraged perturbations at the input [24, 30] and intermediate layers [29] to perform weakly and fully supervised localization.

Approximation-based methods. Black-box models can be analyzed by approximating them (locally) with simpler, more interpretable models. The gradient method of [23] and, more explicitly, LIME [21], do so using linear models. Approximations using decision trees or other models are also possible, although less applicable to visual inputs.

Visualizations of intermediate activations. To characterize a filter's behavior, Zeiler and Fergus [31] show dataset examples from the training set that maximally activate that filter. Similarly, activation maximization [23] learns an input image that maximally activates a filter. Feature inversion [13] learns an image that reconstructs a network's intermediate activations while leveraging a natural image prior for visual clarity. Subsequent works tackled the problem of improving the natural image prior for feature inversion and/or activation maximization [28, 19, 16, 18, 17]. Recently, some methods have measured the performance of single [4, 34] and combinations of [11, 8] filter activations on probe tasks like classification and segmentation to identify which filter(s) encode what concepts.

One difficulty in undertaking channel attribution is that, unlike spatial attribution, where a salient image region is naturally interpretable to humans, simply identifying "important channels" is insufficient as they are not naturally interpretable. To address this, we combine the aforementioned visualization techniques with channel attribution.

3. Method

We first summarize the perturbation analysis of [7] and then introduce our extremal perturbations framework.

3.1. Perturbation analysis

Let $\boldsymbol{x} : \Omega \to \mathbb{R}^3$ be a colour image, where $\Omega = \{0, \dots, H-1\} \times \{0, \dots, W-1\}$ is a discrete lattice, and let Φ be a model, such as a convolutional neural network, that maps the image to a scalar output value $\Phi(\boldsymbol{x}) \in \mathbb{R}$. The latter could be an output activation, corresponding to a class prediction score, in a model trained for image classification, or an intermediate activation.

In the following, we investigate which parts of the input $\boldsymbol{x}$ strongly excite the model, causing the response $\Phi(\boldsymbol{x})$ to be large. In particular, we would like to find a *mask* $\boldsymbol{m}$ assigning to each pixel $u \in \Omega$ a value $\boldsymbol{m}(u) \in \{0, 1\}$, where $\boldsymbol{m}(u) = 1$ means that the pixel strongly contributes to the output and $\boldsymbol{m}(u) = 0$ that it does not.

In order to assess the importance of a pixel, we use the mask to induce a local perturbation of the image, denoted $\hat{\boldsymbol{x}} = \boldsymbol{m} \otimes \boldsymbol{x}$. The details of the perturbation model are discussed below, but for now it suffices to say that pixels for which $\boldsymbol{m}(u) = 1$ are preserved, whereas the others are blurred away. The goal is then to find a small subset of pixels that, when preserved, are sufficient to retain a large value of the output $\Phi(\boldsymbol{m} \otimes \boldsymbol{x})$.

Fong and Vedaldi [7] propose to identify such salient pixels by solving an optimization problem of the type:

$$\boldsymbol{m}_{\lambda,\beta} = \operatorname*{argmax}_{\boldsymbol{m}} \Phi(\boldsymbol{m} \otimes \boldsymbol{x}) - \lambda \|\boldsymbol{m}\|_1 - \beta \mathcal{S}(\boldsymbol{m}). \quad (1)$$

The first term encourages the network's response to be large. The second encourages the mask to select a small part of the input image, blurring as many pixels as possible. The third further regularises the smoothness of the mask by penalising irregular shapes.

The problem with this formulation is that the meaning of the trade-off established by optimizing eq. (1) is unclear as the three terms, model response, mask area and mask regularity, are not commensurate. In particular, choosing different λ and β values in eq. (1) will result in different masks without a clear way of comparing them.

3.2. Extremal perturbations

In order to remove the balancing issues with eq. (1), we propose to constrain the area of the mask to a fixed value (as a fraction $a|\Omega|$ of the input image area). Furthermore, we control the smoothness of the mask by choosing it in a fixed set $\mathcal{M}$ of sufficiently smooth functions. Then, we find

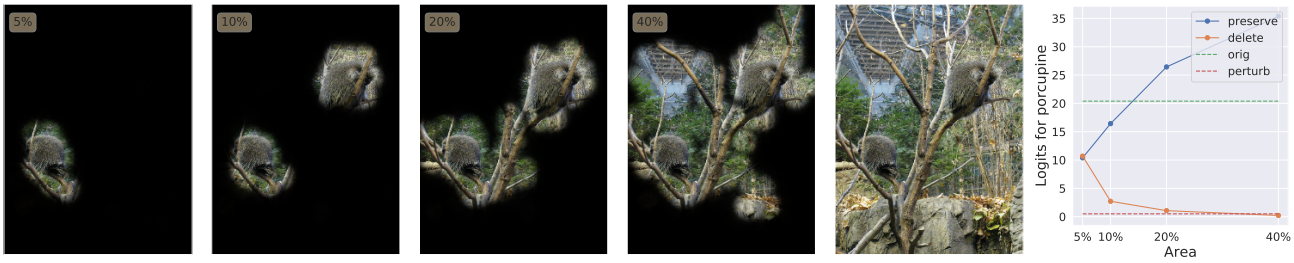

Figure 3: **Extremal perturbations and monotonic effects.** Left: "porcupine" masks computed for several areas a (a in box). Right: $\Phi(\boldsymbol{m}_a \otimes \boldsymbol{x})$ (preservation; blue) and $\Phi((1-\boldsymbol{m}_a)\otimes\boldsymbol{x})$ (deletion; orange) plotted as a function of a. At $a \approx 15\%$ the preserved region scores *higher* than preserving the entire image (green). At $a \approx 20\%$, perturbing the complementary region scores *similarly* to fully perturbing the entire image (red).

the mask of that size that maximizes the model's output:

$$\boldsymbol{m}_a = \operatorname*{argmax}_{\boldsymbol{m}:\ \|\boldsymbol{m}\|_1 = a|\Omega|,\ \boldsymbol{m}\in\mathcal{M}} \Phi(\boldsymbol{m}\otimes\boldsymbol{x}). \tag{2}$$

Note that the resulting mask is a function of the chosen area a only. With this, we can define the concept of *extremal perturbation* as follows. Consider a lower bound Φ_0 on the model's output (for example we may set $\Phi_0 = \tau\Phi(\boldsymbol{x})$ to be a fraction τ of the model's output on the unperturbed image). Then, we search for the *smallest mask* that achieves at least this output level. This amounts to sweeping the area parameter a in eq. (2) to find

$$a^* = \min\{a : \Phi(\boldsymbol{m}_a\otimes\boldsymbol{x}) \geq \Phi_0\}. \tag{3}$$

The mask $\boldsymbol{m}_{a^*}$ is extremal because preserving a smaller portion of the input image is not sufficient to excite the network's response above Φ_0. This is illustrated in fig. 3.

Interpretation. An extremal perturbation is a single mask $\boldsymbol{m}_{a^*}$ that results in a large model response, in the sense that $\Phi(\boldsymbol{m}_{a^*}\otimes\boldsymbol{x}) \geq \Phi_0$. However, due to extremality, we *also* know that any smaller mask does not result in an equally large response: $\forall \boldsymbol{m}:\ \|\boldsymbol{m}\|_1 < \|\boldsymbol{m}_{a^*}\|_1 \ \Rightarrow\ \Phi(\boldsymbol{m}\otimes\boldsymbol{x}) < \Phi_0$. Hence, a single extremal mask is informative because it characterises a *whole family* of input perturbations.

This connects extremal perturbations to methods like [21, 7], which explain a network by finding a succinct and interpretable description of its input-output mapping. For example, the gradient [23] and LIME [21] approximate the network locally around an input $\boldsymbol{x}$ using the Taylor expansion $\Phi(\boldsymbol{x}') \approx \langle\nabla\Phi(\boldsymbol{x}), \boldsymbol{x}'-\boldsymbol{x}\rangle + \Phi(\boldsymbol{x})$; their explanation is the gradient $\nabla\Phi(\boldsymbol{x})$ and their perturbations span a neighbourhood of $\boldsymbol{x}$.

Preservation vs deletion. Formulation (2) is analogous to what [7] calls the "preservation game" as the goal is to find a mask that preserves (maximises) the model's response. We also consider their "deletion game" obtaining by optimising $\Phi((1-\boldsymbol{m})\otimes\boldsymbol{x})$ in eq. (2), so that the goal is to suppress the response when looking outside the mask, and the hybrid [5], obtained by optimising $\Phi(\boldsymbol{m}\otimes\boldsymbol{x}) - \Phi((1-\boldsymbol{m})\otimes\boldsymbol{x})$, where the goal is to simultaneously preserve the response inside the mask and suppress it outside

3.3. Area constraint

Enforcing the area constraint in eq. (2) is non-trivial; here, we present an effective approach to do so (other approaches like [10] do not encourage binary masks). First, since we would like to optimize eq. (2) using a gradient-based method, we relax the mask to span the full range $[0, 1]$. Then, a possible approach would be to count how many values $\boldsymbol{m}(u)$ are sufficiently close to the value 1 and penalize masks for which this count deviates from the target value $a|\Omega|$. However, this approach requires soft-counting, with a corresponding tunable parameter for binning.

In order to avoid such difficulties, we propose instead to *vectorize and sort* in non-decreasing order the values of the mask $\boldsymbol{m}$, resulting in a vector $\operatorname{vecsort}(\boldsymbol{m}) \in [0,1]^{|\Omega|}$. If the mask $\boldsymbol{m}$ satisfies the area constraint exactly, then the output of $\operatorname{vecsort}(\boldsymbol{m})$ is a vector $\mathbf{r}_a \in [0,1]^{|\Omega|}$ consisting of $(1-a)|\Omega|$ zeros followed by $a|\Omega|$ ones. This is captured by the regularization term: $R_a(\boldsymbol{m}) = \|\operatorname{vecsort}(\boldsymbol{m}) - \mathbf{r}_a\|^2$. We can then rewrite eq. (2) as

$$\boldsymbol{m}_a = \operatorname*{argmax}_{\boldsymbol{m}\in\mathcal{M}} \Phi(\boldsymbol{m}\otimes\boldsymbol{x}) - \lambda R_a(\boldsymbol{m}). \tag{4}$$

Note that we have reintroduced a weighting factor λ in the formulation, so on a glance we have lost the advantage of formulation (2) over the one of eq. (1). In fact, this is not the case: during optimization we simply set λ to be as large as numerics allow it as we expect the area constraint to be (nearly) exactly satisfied; similarly to a logarithmic barrier, λ then has little effect on which mask $\boldsymbol{m}_a$ is found.

3.4. Perturbation operator

In this section we define the perturbation operator $\boldsymbol{m}\otimes\boldsymbol{x}$. To do so, consider a *local perturbation operator* $\pi(\boldsymbol{x}; u, \sigma) \in \mathbb{R}^3$ that applies a perturbation of intensity $\sigma \geq 0$ to pixel $u \in \Omega$. We assume that the lowest intensity $\sigma = 0$ corresponds to no perturbation, i.e. $\pi(\boldsymbol{x}; u, 0) =$

$\boldsymbol{x}(u)$. Here we use as perturbations the Gaussian blur[2]

$$\pi_g(\boldsymbol{x}; u, \sigma) = \frac{\sum_{v\in\Omega} g_\sigma(u-v)\boldsymbol{x}(v)}{\sum_{v\in\Omega} g_\sigma(u-v)}, \quad g_\sigma(u) = e^{-\frac{\|u\|^2}{2\sigma^2}}.$$

The mask $\boldsymbol{m}$ then doses the perturbation spatially: $(\boldsymbol{m}\otimes\boldsymbol{x})(u) = \pi(\boldsymbol{x}; u, \sigma_{\max}\cdot(1-\boldsymbol{m}(u)))$ where $\sigma_{\max}$ is the maximum perturbation intensity.[3]

3.5. Smooth masks

Next, we define the space of smooth masks $\mathcal{M}$. For this, we consider an auxiliary mask $\bar{\boldsymbol{m}} : \Omega \to [0,1]$. Given that the range of $\bar{\boldsymbol{m}}$ is bounded, we can obtain a smooth mask $\boldsymbol{m}$ by convolving $\bar{\boldsymbol{m}}$ by a Gaussian or similar kernel $\boldsymbol{k} : \Omega \to \mathbb{R}_+$[4] via the typical *convolution operator*:

$$\boldsymbol{m}(u) = Z^{-1}\sum_{v\in\Omega} \boldsymbol{k}(u-v)\bar{\boldsymbol{m}}(v) \tag{5}$$

where Z normalizes the kernel to sum to one. However, this has the issue that setting $\bar{\boldsymbol{m}}(u) = 1$ does not necessarily result in $\boldsymbol{m}(u) = 1$ after filtering, and we would like our final mask to be (close to) binary.

To address this issue, we consider the *max-convolution operator*:

$$\boldsymbol{m}(u) = \max_{v\in\Omega} \boldsymbol{k}(u-v)\bar{\boldsymbol{m}}(v). \tag{6}$$

This solves the issue above while at the same time guaranteeing that the smoothed mask does not change faster than the smoothing kernel, as shown in the following lemma (proof in supp. mat.).

Lemma 1. *Consider functions $\bar{\boldsymbol{m}}, \boldsymbol{k} : \Omega \to [0,1]$ and let $\boldsymbol{m}$ be defined as in eq. (6). If $\boldsymbol{k}(0) = 1$, then $\bar{\boldsymbol{m}}(u) \le \boldsymbol{m}(u) \le 1$ for all $u \in \Omega$; in particular, if $\bar{\boldsymbol{m}}(u) = 1$, then $\boldsymbol{m}(u) = 1$ as well. Furthermore, if $\boldsymbol{k}$ is Lipschitz continuous with constant K, then $\boldsymbol{m}$ is also Lipschitz continuous with a constant at most as large as K.*

The max operator in eq. (6) yields sparse gradients. Thus, to facilitate optimization, we introduce the *smooth max* operator[5], smax, to replace the max operator. For a function $f(u), u \in \Omega$ and temperature $T > 0$:

$$\operatorname*{smax}_{u\in\Omega;T} f(u) = \frac{\sum_{u\in\Omega} f(u)\exp f(u)/T}{\sum_{u\in\Omega}\exp f(u)/T} \tag{7}$$

[2]Another choice is the fade-to-black perturbation which, for $0 \le \sigma \le 1$, is given by $\pi_f(\boldsymbol{x}; u, \sigma) = (1-\sigma)\cdot\boldsymbol{x}(u)$.

[3]For efficiency, this is implemented by generating a *perturbation pyramid* $\pi(\boldsymbol{x}; \cdot, \sigma_{\max}\cdot l/L)$, $l = 0, \dots, L$ that contains $L+1$ progressively more perturbed versions of the image. Then $\boldsymbol{m}\otimes\boldsymbol{x}$ can be computed via bilinear interpolation by using $(u, \boldsymbol{m}(u))$ as an indices in the pyramid.

[4]It is easy to show that in this case the derivative of the smoothed mask $\||\nabla(\boldsymbol{k} * \bar{\boldsymbol{m}})\|\| \le \|\nabla\boldsymbol{k}\|$ is always less than the one of the kernel.

[5]Not to be confused with the softmax with temperature, as in [9].

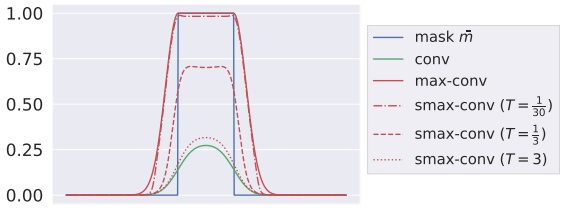

Figure 4: **Convolution operators for smooth masks.** Gaussian smoothing a mask (blue) with the typical convolution operator yields a dampened, smooth mask (green). Our max-convolution operator mitigates this effect while still smoothing (red solid). Our smax operator, which yields more distributed gradients than max, varies between the other two convolution operators (red dotted).

The smax operator smoothly varies from behaving like the mean operator in eq. (5) as $T \to \infty$ to behaving like the max operator as $T \to 0$ (see fig. 4). This operator is used instead of max in eq. (6).

Implementation details. In practice, we use a smaller parameterization mask $\bar{\boldsymbol{m}}$ defined on a lattice $\bar{\Omega} = \{0,\dots,\bar{H}-1\}\times\{0,\dots,\bar{W}-1\}$, where the full-resolution mask $\boldsymbol{m}$ has dimensions $H = \rho\bar{H}$ and $W = \rho\bar{W}$. We then modify (6) to perform upsampling in the same way as the standard convolution transpose operator.

4. Experiments

Implementation details. Unless otherwise noted, all visualizations use the ImageNet validation set, the VGG16 network and the preservation formulation (Sec. 3.2). Specifically, $\Phi(\boldsymbol{x})$ is the classification score (before softmax) that network associates to the ground-truth class in the image. Masks are computed for areas $a \in \{0.05, 0.1, 0.2, 0.4, 0.6, 0.8\}$. To determine the optimal area a^* of the extremal perturbations (3), we set the threshold $\Phi_0 = \Phi(\boldsymbol{x})$ (which is the score on the unperturbed image).

Masks are optimised using SGD, initializing them with all ones (everything preserved). SGD uses momentum 0.9 and 1600 iterations. λ is set to 300 and doubled at 1/3 and 2/3 of the iterations and, in eq. (7), $1/T \approx 20$. Before upsampling, the kernel $\boldsymbol{k}(u) = k(\|u\|)$ is a radial basis function with profile $k(z) = \exp\left(\max\{0, z-1\}^2/4\right)$, chosen so that neighbour disks are centrally flat and then decay smoothly.

4.1. Qualitative comparison

Figure 2 shows a qualitative comparison between our method and others. We see that our criterion of $\Phi_0 = \Phi(x)$ chooses fairly well-localized masks in most cases. Masks tend to cover objects tightly, are sharp, and clearly identify a region of interest in the image. Figure 5 shows what the network considered to be most discriminative ($a = 5\%$; e.g.,

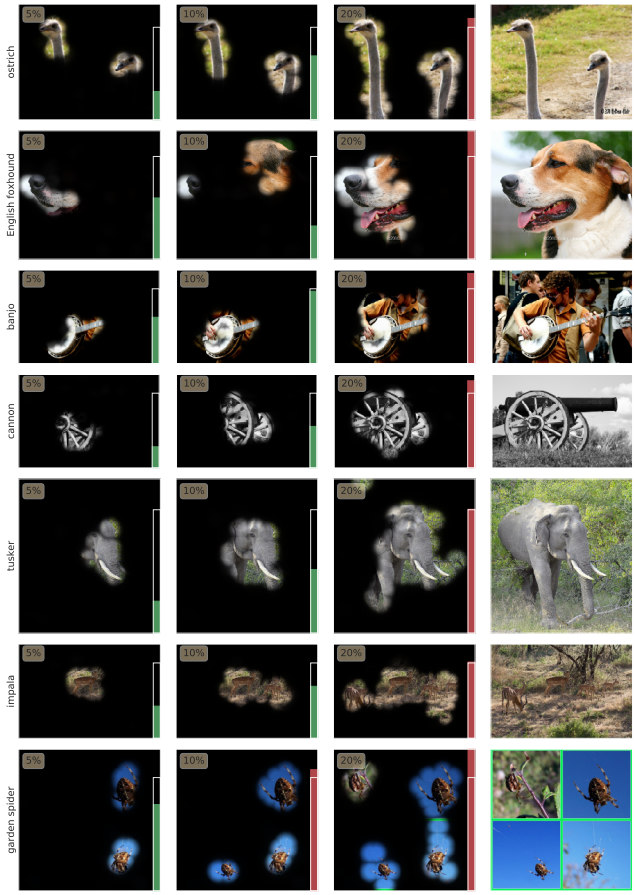

Figure 5: **Area growth.** Although each mask is learned independently, these plots highlight what the network considers to be most discriminative and complete. The bar graph visualizes $\Phi(\boldsymbol{m}_a \odot \boldsymbol{x})$ as a normalized fraction of $\Phi_0 = \Phi(\boldsymbol{x})$ (and saturates after exceeding Φ_0 by 25%).

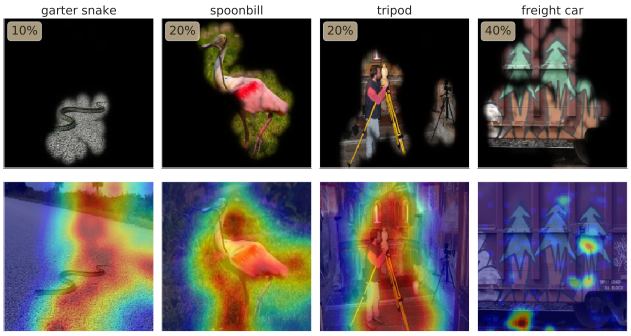

Figure 6: **Comparison with [7].** Our extremal perturbations (top) vs. masks from Fong and Vedaldi [7] (bottom).

banjo fret board, elephant tusk) and complete ($a = 20\%$) as the area increases. We notice that evidence from several objects accumulates monotonically (e.g., impala and spider) and that foreground (e.g., ostrich) or discriminative parts (e.g., dog's nose) are usually sufficient.

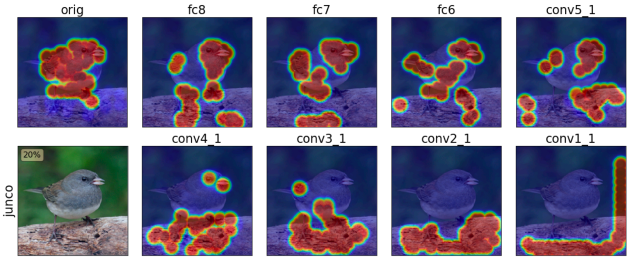

Figure 7: **Sanity check [2].** Model weights are progressively randomized from fc8 to conv1_1 in VGG16, demonstrating our method's sensitivity to model weights.

	VOC07 Test (All/Diff)		*COCO14 Val (All/Diff)*	
Method	*VGG16*	*ResNet50*	*VGG16*	*ResNet50*
Cntr.	69.6/42.4	69.6/42.4	27.8/19.5	27.8/19.5
Grad	76.3/56.9	72.3/56.8	37.7/31.4	35.0/29.4
DConv	67.5/44.2	68.6/44.7	30.7/23.0	30.0/21.9
Guid.	75.9/53.0	77.2/59.4	39.1/31.4	42.1/35.3
MWP	77.1/56.6	84.4/70.8	39.8/32.8	49.6/43.9
cMWP	79.9/66.5	**90.7**/82.1	49.7/44.3	**58.5/53.6**
RISE*	86.9/75.1	86.4/78.8	50.8/45.3	54.7/50.0
GCAM	86.6/74.0	90.4/**82.3**	**54.2/49.0**	57.3/52.3
Ours*	**88.0/76.1**	88.9/78.7	51.5/45.9	56.5/51.5

Table 1: **Pointing game.** Mean accuracy on the pointing game over the full data splits and a subset of difficult images (defined in [32]). Results from PyTorch re-implementation using TorchRay package (* denotes average over 3 runs).

In fig. 6, we compare our masks to those of Fong and Vedaldi [7]. The stability offered by controlling the area of the perturbation is obvious in these examples. Lastly, we visualize a sanity check proposed in Adebayo et al. [2] in fig. 7 (we use the "hybrid" formulation). Unlike other backprop-based methods, our visualizations become significantly different upon weight randomization (see supp. mat. for more qualitative examples).

4.2. Pointing game

A common approach to evaluate attribution methods is to correlate their output with semantic annotations in images. Here we consider in particular the pointing game of Zhang et al. [32]. For this, an attribution method is used to compute a saliency map for each of the object classes present in the image. One scores a hit if the maximum point in the saliency map is contained within the object; The overall accuracy is the number of hits over number of hits plus misses.

Table 1 shows results for this metric and compares our method against the most relevant work in the literature on PASCAL VOC [6] (using the 2007 test set of 4952 images) and COCO [12] (using the 2014 validation set of $\approx 50k$ im-

ages). We see that our method is competitive with VGG16 and ResNet50 networks. In contrast, Fong and Vedaldi's [7] was not competitive in this benchmark (although they reported results using GoogLeNet).

Implementation details. Since our masks are binary, there is no well defined maximum point. To apply our method to the pointing game, we thus run it for areas $\{0.025, 0.05, 0.1, 0.2\}$ for PASCAL and $\{0.018, 0.025, 0.05, 0.1\}$ for COCO (due to the smaller objects in this dataset). The binary masks are summed and a Gaussian filter with standard deviation equal to 9% of the shorter side of the image applied to the result to convert it to a saliency map. We use the original Caffe models of [32] converted to PyTorch and use the preservation formulation of our method.

4.3. Monotonicity of visual evidence

Eq. (2) implements the "preservation game" and searches for regions of a given area that *maximally activate* the networks' output. When this output is the confidence score for a class, we hypothesise that hiding evidence from the network would only make the confidence lower, i.e., we would expect the effect of maximal perturbations to be ordered consistently with their size:

$$a_1 \leq a_2 \;\Rightarrow\; \Phi(\boldsymbol{m}_{a_1} \otimes \boldsymbol{x}) \leq \Phi(\boldsymbol{m}_{a_2} \otimes \boldsymbol{x}) \qquad (8)$$

However, this may not always be the case. In order to quantify the frequency of this effect, we test whether eq. (8) holds for all $a_1, a_2 < a^*$, where a^* is the optimal area of the extremal perturbation (eq. (3), where $\Phi_0 = \Phi(\boldsymbol{x})$). Empirically, we found that this holds for 98.45% of ImageNet validation images, which indicates that evidence is in most cases integrated monotonically by the network.

More generally, our perturbations allow us to sort and investigate how information is integrated by the model in order of importance. This is shown in several examples in fig. 5 where, as the area of the mask is progressively increased, different parts of the objects are prioritised.

5. Attribution at intermediate layers

Lastly, we extend extremal perturbations to the *direct* study of the intermediate layers in neural networks. This allows us to highlight a novel use case of our area loss and introduce a new technique for understanding which channels are salient for classification.

As an illustration, we consider in particular channel-wise perturbations. Let $\Phi_l(x) \in \mathbb{R}^{K_l \times H_l \times W_l}$ be the intermediate representation computed by a neural network Φ up to layer l and let $\Phi_{l+} : \mathbb{R}^{K_l \times H_l \times W_l} \to \mathbb{R}$ represent the rest of model, from layer l to the last layer. We then re-formulate the preservation game from eq. (4) as:

$$\boldsymbol{m}_a = \operatorname*{argmax}_{\boldsymbol{m}} \Phi_{l+}(\boldsymbol{m} \otimes \Phi_l(\boldsymbol{x})) - \lambda R_a(\boldsymbol{m}). \qquad (9)$$

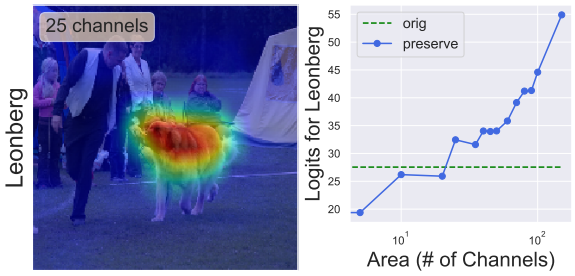

Figure 8: **Attribution at intermediate layers.** Left: This is visualization (eq. (11)) of the optimal channel attribution mask $\boldsymbol{m}_{a^*}$, where $a^* = 25$ channels, as defined in eq. (10). Right: This plot shows that class score monotonically increases as the area (as the number of channels) increases.

Here, the mask $\boldsymbol{m} \in [0,1]^{K_l}$ is a vector with one element per channel which element-wise multiplies with the activations $\Phi_l(\boldsymbol{x})$, broadcasting values along the spatial dimensions. Then, the extremal perturbation $\boldsymbol{m}_{a^*}$ is selected by choosing the optimal area

$$a^* = \min\{a : \Phi_{l+}(\boldsymbol{m}_a \otimes \Phi_l(\boldsymbol{x})) \geq \Phi_0\}. \qquad (10)$$

We assume that the output Φ_{l+} is the pre-softmax score for a certain image class and we set the $\Phi_0 = \Phi(\boldsymbol{x})$ to be the model's predicted value on the unperturbed input (fig. 8).

Implementation details. In these experiments, we use GoogLeNet [27] and focus on layer l =`inception4d`, where $H_l = 14, W_l = 14, K_l = 528$. We optimize eq. (9) for 300 iterations with a learning rate of 10^{-2}. The parameter λ linearly increases from 0 to 1500 during the first 150 iterations, after which $\lambda = 1500$ stays constant. We generate channel-wise perturbation masks for areas $a \in \{1, 5, 10, 20, 25, 35, 40, 50, 60, 70, 80, 90, 100, 150, 200, 250, 300, 350, 400, 450, 528\}$, where a denotes the number of channels preserved.

The saliency heatmaps in fig. 8 and fig. 9 for channel-wise attribution are generated by summing over the channel dimension the element-wise product of the channel attribution mask and activation tensor at layer l:

$$\boldsymbol{v} = \sum_{k \in K} \boldsymbol{m}^k_{a^*} \otimes \Phi^k_l(\boldsymbol{x}) \qquad (11)$$

5.1. Visualizing per-instance channel attribution

Unlike per-instance input-level spatial attribution, which can be visualized using a heatmap, per-instance intermediate channel attribution is more difficult to visualize because simply identifying important channels is not necessarily human-interpretable. To address this problem, we use feature inversion [15, 19] to find an image that maximises the dot product of the channel attribution vector and the activation tensor (see [19] for more details):

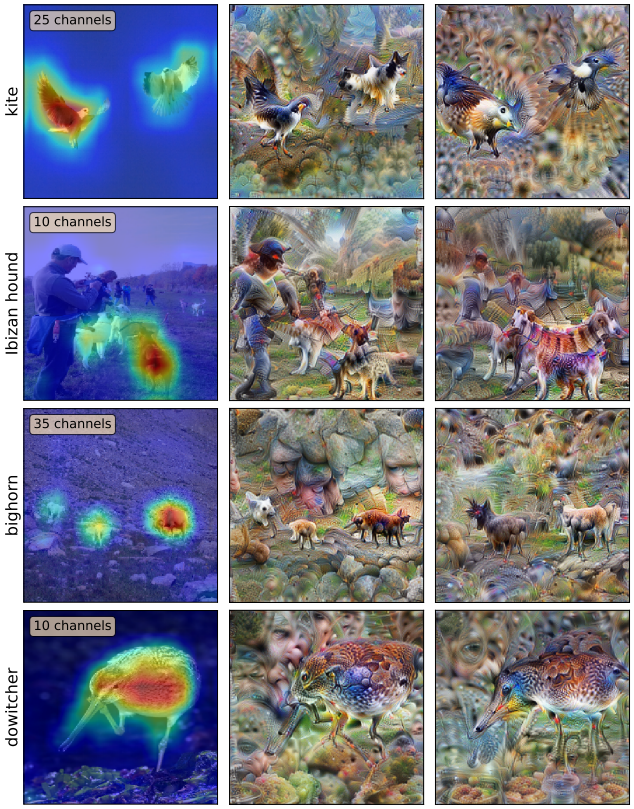

Figure 9: **Per-instance channel attribution visualization.** Left: input image overlaid with channel saliency map (eq. (11)). Middle: feature inversion of original activation tensor. Right: feature inversion of activation tensor perturbed by optimal channel mask $\boldsymbol{m}_{a^*}$. By comparing the difference in feature inversions between un-perturbed (middle) and perturbed activations (right), we can identify the salient features that our method highlights.

$$I^* = \underset{I}{\operatorname{argmax}} \{(\boldsymbol{m}_{a^*} \otimes \Phi_l(\boldsymbol{x})) \cdot \Phi_l(I)\} \tag{12}$$

where $\boldsymbol{m}_a^*$ is optimal channel attribution mask at layer l for input image $\boldsymbol{x}$ and $\Phi_l(I)$ is the activation tensor at layer l for image I, the image we are learning.

This inverted image allows us to identify the parts of the input image that are salient for a particular image to be correctly classified by a model. We can compare the feature inversions of activation tensors perturbed with channel mask (right column in fig. 9) to the inversions of original, unperturbed activation tensors (middle column) to get a clear idea of the most discriminative features of an image.

Since the masks are roughly binary, multiplying $\boldsymbol{m}_a^*$ with the activation tensor $\Phi_l(\boldsymbol{x})$ in eq. (12) zeroes out non-salient channels. Thus, the differences in the feature inversions of original and perturbed activations in fig. 9 highlight regions encoded by salient channels identified in our attribution masks (i.e., the channels that are not zeroed out in eq. (12)).

Figure 10: **Discovery of salient, class-specific channels.** By analyzing $\bar{\boldsymbol{m}}_c$, the average over all $\boldsymbol{m}_{a^*}$ for class c (see Sec. 5.2), we automatically find salient, class-specific channels like these. First column: channel feature inversions; all others: dataset examples.

5.2. Visualizing per-class channel attribution

We can also use channel attribution to identify important, class-specific channels. In contrast to other methods, which explicitly aim to find class-specific channels and/or directions at a global level [8, 11, 34], we are able to similarly do so "for free" using only our per-instance channel attribution masks. After estimating an optimal masks $\boldsymbol{m}_{a^*}$ for all ImageNet validation images, we then create a per-class attribution mask $\bar{\boldsymbol{m}}_c \in [0,1]^K$ by averaging the optimal masks of all images in a given class c. Then, we can identify the most important channel for a given class as follows: $k_c^* = \operatorname{argmax}_{k \in K} \bar{\boldsymbol{m}}_c^k$. In fig. 10, we visualize two such channels via feature inversions. Qualitatively, these feature inversions of channels k_c^* are highly class-specific.

6. Conclusion

We have introduced the framework of extremal perturbation analysis, which avoids some of the issues of prior work that use perturbations to analyse neural networks. We have also presented a few technical contributions to compute such extremal perturbation. Among those, the rank-order area constraint can have several other applications in machine learning beyond the computation of extremal perturbations. We have extended the perturbations framework to perturbing intermediate activations and used this to explore a number of properties of the representation captured by a model. In particular, we have visualized, likely for the first time, the difference between perturbed and unperturbed activations using a representation inversion technique. Lastly, we released TorchRay [1], a PyTorch interpretability library in which we've re-implemented popular methods and benchmarks to encourage reproducible research.

Acknowledgements. We are grateful for support from the Open Philanthropy Project (R.F.), the Rhodes Trust (M.P.), and EPSRC EP/L015897/1 (CDT in Autonomous Intelligent Machines and Systems) (M.P). We also thank Jianming Zhang and Samuel Albanie for help on re-implementing the Pointing Game [32] in PyTorch.

References

[1] Torchray. github.com/facebookresearch/TorchRay, 2019. 8

[2] Julius Adebayo, Justin Gilmer, Ian Goodfellow, Moritz Hardt, and Been Kim. Sanity checks for saliency maps. In *Proc. NeurIPS*, 2018. 3, 6

[3] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015. 1, 3

[4] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *Proc. CVPR*, 2017. 3

[5] Piotr Dabkowski and Yarin Gal. Real time image saliency for black box classifiers. In *Proc. NeurIPS*, 2017. 1, 3, 4

[6] Mark Everingham, SM Ali Eslami, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *IJCV*, 111(1):98–136, Jan. 2015. 6

[7] Ruth Fong and Andrea Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. In *Proc. ICCV*, 2017. 1, 2, 3, 4, 6, 7

[8] Ruth Fong and Andrea Vedaldi. Net2Vec: Quantifying and explaining how concepts are encoded by filters in deep neural networks. In *Proc. CVPR*, 2018. 3, 8

[9] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv*, 2015. 5

[10] Hoel Kervadec, Jose Dolz, Meng Tang, Eric Granger, Yuri Boykov, and Ismail Ben Ayed. Constrained-cnn losses for weakly supervised segmentation. *Medical image analysis*, 54:88–99, 2019. 4

[11] Been Kim, Martin Wattenberg, Justin Gilmer, Carrie Cai, James Wexler, Fernanda Viegas, and Rory Sayres. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). In *Proc. ICML*, 2017. 3, 8

[12] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Proc. ECCV*, 2014. 6

[13] Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *Proc. CVPR*, 2015. 2, 3

[14] Aravindh Mahendran and Andrea Vedaldi. Salient deconvolutional networks. In *Proc. ECCV*, 2016. 3

[15] Aravindh Mahendran and Andrea Vedaldi. Visualizing deep convolutional neural networks using natural pre-images. *IJCV*, 2016. 7

[16] Alexander Mordvintsev, Nicola Pezzotti, Ludwig Schubert, and Chris Olah. Differentiable image parameterizations. *Distill*, 3(7):e12, 2018. 2, 3

[17] Anh Nguyen, Jeff Clune, Yoshua Bengio, Alexey Dosovitskiy, and Jason Yosinski. Plug & play generative networks: Conditional iterative generation of images in latent space. In *Proc. CVPR*, 2017. 3

[18] Anh Nguyen, Alexey Dosovitskiy, Jason Yosinski, Thomas Brox, and Jeff Clune. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. In *Proc. NeurIPS*, 2016. 3

[19] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2(11):e7, 2017. 2, 3, 7

[20] Vitali Petsiuk, Abir Das, and Kate Saenko. Rise: Randomized input sampling for explanation of black-box models. In *Proc. BMVC*, 2018. 1, 2, 3

[21] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why should I trust you?" explaining the predictions of any classifier. In *Proc. KDD*, 2016. 3, 4

[22] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-CAM: Visual explanations from deep networks via gradient-based localization. In *Proc. ICCV*, 2017. 1, 2, 3

[23] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *Proc. ICLR workshop*, 2014. 1, 2, 3, 4

[24] Krishna Kumar Singh and Yong Jae Lee. Hide-and-seek: Forcing a network to be meticulous for weakly-supervised object and action localization. In *Proc. ICCV*, 2017. 3

[25] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise. *arXiv*, 2017. 3

[26] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. 2015. 1, 2, 3

[27] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proc. CVPR*, 2015. 7

[28] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Deep image prior. In *Proc. CVPR*, 2018. 2, 3

[29] Xiaolong Wang, Abhinav Shrivastava, and Abhinav Gupta. A-fast-rcnn: Hard positive generation via adversary for object detection. In *Proc. CVPR*, 2017. 3

[30] Yunchao Wei, Jiashi Feng, Xiaodan Liang, Ming-Ming Cheng, Yao Zhao, and Shuicheng Yan. Object region mining with adversarial erasing: A simple classification to semantic segmentation approach. In *Proc. CVPR*, 2017. 3

[31] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Proc. ECCV*, 2014. 1, 3

[32] Jianming Zhang, Sarah Adel Bargal, Zhe Lin, Jonathan Brandt, Xiaohui Shen, and Stan Sclaroff. Top-down neural attention by excitation backprop. *International Journal of Computer Vision*, 126(10):1084–1102, 2018. 1, 3, 6, 7, 8

[33] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proc. CVPR*, 2016. 3

[34] Bolei Zhou, Yiyou Sun, David Bau, and Antonio Torralba. Revisiting the importance of individual units in cnns via ablation. *arXiv*, 2018. 3, 8

A. Implementation details

A.1. Generating smooth masks

We implement the equation:

$$\hat{m}(u) = \text{pool}_i\, g(u - u_i) m_i$$

Here $i = 0, \dots, N-1$ are samples of the input mask, $u = 0, \dots, W-1$ samples ouf the output mask, and u_i is the mapping between input and output samples, $u_i = ai + b$. We assume that the kernel k has a "radius" σ, in the sense that only samples $|u - u_i| \leq \sigma$ matter.

In order to compute this expansion fast, we use the unpool operator. In order to do so, unpool is applied to m with window $K = 2R + 1$ and padding P. This results in the signal

$$m'_{k,i} = m_{i+k-P}, \quad 0 \leq i \leq W' - 1, \quad 0 \leq k \leq K - 1, \qquad W' = N - K + 2P + 1.$$

We then use nearest-neighbour upsampling in order to bring this signal in line with the resolution of the output:

$$m''_{k,u} = m_{\lfloor \frac{u}{s} \rfloor + k - P}, \quad 0 \leq u \leq W'' - 1, \quad 0 \leq k \leq K - 1.$$

Here the upsampling factor is given by $s = W''/W'$. In PyTorch, we specify upsampling via the input size W' and the output size W'', so we need to choose W'' appropriately.

To conclude, we do so as follows. We choose a σ (kernel width in image pixels) and and s (kernel step in pixels). We also choose a margin $b \geq 0$ to avoid border effects and set $a = s$. With this, we see that computing $\hat{m}(u)$ requires samples:

$$u - \sigma \leq u_i \leq u + \sigma \quad \Leftrightarrow \quad \frac{u}{s} - \frac{\sigma + b}{s} \leq i \leq \frac{u}{s} + \frac{\sigma - b}{s}.$$

On the other hand, at location u in $m''_{k,u}$ we have pooled samples m_i for which:

$$\left\lfloor \frac{u}{s} \right\rfloor - P \leq i \leq \left\lfloor \frac{u}{s} \right\rfloor + K - 1 - P.$$

Hence we require

$$\left\lfloor \frac{u}{s} \right\rfloor - P \leq \frac{u}{s} - \frac{\sigma + b}{s} \quad \Rightarrow \quad P \geq \frac{\sigma + b}{s} + \left\lfloor \frac{u}{s} \right\rfloor - \frac{u}{s}.$$

Conservatively, we take:

$$P = 1 + \left\lceil \frac{\sigma + b}{s} \right\rceil$$

The other bound is:

$$\frac{u}{s} + \frac{\sigma - b}{s} \leq \left\lfloor \frac{u}{s} \right\rfloor + K - 1 - P.$$

Hence:

$$K \geq \frac{u}{s} - \left\lfloor \frac{u}{s} \right\rfloor + \frac{\sigma - b}{s} + P + 1$$

Hence, conservatively we take:

$$K = 3 + \left\lceil \frac{\sigma + b}{s} \right\rceil + \left\lceil \frac{\sigma - b}{s} \right\rceil.$$

Since $K = 2R + 1$ and $b \approx \sigma$, we set

$$R = 1 + \left\lceil \frac{\sigma}{s} \right\rceil.$$

In this way, we obtain a pooled mask:

$$\bar{m}(u) = \underset{i}{\text{pool}}\, g(u - u_i)\, m_i = \underset{0 \leq k \leq K-1}{\text{pool}}\, g_{k,u} m''_{k,u},$$

where

$$g_{k,u} = g(u - \bar{u}(u,k)), \quad \bar{u}(u,k) = \left\lfloor \frac{u}{s} \right\rfloor + k - P.$$

Hence, the steps are: given the input mask parameters m_i, use unpooling to obtain $m'_{k,i}$ and then upsampling to obtain $m''_{k,u}$. Then use the equation above to pool using a precomputed weights $g_{k,u}$.

Generally, the input to the mask generator are: s, σ and the desired mask $\hat{m}(u)$ width W. So far, we have obtained a mask $\bar{m}(u)$ with with W'', where $W'' = sW'$ is chosen to obtain the correct scaling factor and $W' = N - K + 2P + 1$. As a rule of thumb, we set $N = \lceil W/s \rceil$ in order to spread the N samples at regular interval over the full stretch W. We then set R, K, P, W' and W'' according to the formulas above. Once $\bar{m}(u)$ is obtained, we take a W-sized crop shifted by b pixels to obtain the final mask $\hat{m}(u)$.

B. Supplementary Materials

The full supplementary materials for this paper can be found at ruthcfong.github.io/files/fong19_extremal_supps.pdf.