Explaining Neural Networks via Perturbing Important Learned Features

Ashkan Khakzar[1] Soroosh Baselizadeh [1] Saurabh Khanduja[1] Seong Tae Kim[1] Nassir Navab[1,2]
[1]Technical University of Munich, Germany [2]Johns Hopkins University, USA
{ashkan.khakzar;soroosh.baselizadeh;saurabh.khanduja;seongtae.kim;nassir.navab}@tum.de

Abstract

Attributing the output of a neural network to the contribution of given input elements is one way of shedding light on the black box nature of neural networks. We propose a novel input feature attribution method that finds an input perturbation that maximally changes the output neuron by exclusively perturbing important hidden neurons (i.e. learned features) on the path to output neuron. Given an input, this is achieved by 1) pruning unimportant neurons, and subsequently 2) finding a local input perturbation that maximizes the output in the pruned network. Since our method considers the importance of hidden neurons (high-level features), it inherently considers interdependencies between multiple input elements, which is vital for input feature attribution. We propose PruneGrad, an efficient gradient-based solution for the pruning and perturbation steps of our method. The efficacy of our method is evaluated by quantitatively benchmarking against other attribution methods using 1) sanity checks, 2) pixel perturbation, and 3) Remove and Retrain (ROAR). Our results show that while most of the existing attribution methods are prone to fail or get mediocre results in at least one benchmark, our proposed method achieves state of the art results in all three benchmarks. The results are further supported by comparative visual evaluation.

1. Introduction

The need for understanding the black-box nature of neural networks has spawned various approaches in interpreting these models. Among them, a family of methods known as input attribution methods explain neural networks by attributing the output of a neural network to the given input's individual elements (e.g. pixels of an image). In other words, they assign an attribution (importance) score to each input element. Some attribution methods [25, 5] derive these importance scores using the local sensitivity of the model to the variation of the input's elements. Another group of attribution methods [4, 24, 28, 31, 14, 3] adopt a more global approach by defining importance relative to a reference (baseline) input. For a given input, the importance score to each input element is assigned by considering the relative contribution of the input elements to the output change. Some reference-based methods [4, 24, 28] use model gradients with custom backpropagation rules. Other reference-based methods [31, 14, 3] use perturbation of input to the reference value and observe the change of output. This effect is studied either by singly removing one element [31] or analyzing the effect of that element's removal in all possible combinations of the elements (i.e. finding Shapley values) [14, 3].

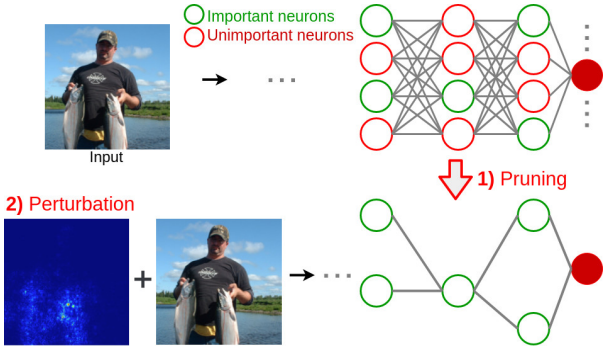


Figure 1: **Overview of our proposed method:** Given an input, 1) prune unimportant neurons (neurons that their removal minimally affects the output of the target neuron) and 2) subsequently find input perturbation that maximally changes the output in the pruned network. The resulting perturbation serves as an explanation of input features that contribute the most to the network's output

However, features in the input are usually composed of multiple elements and the importance of the feature could not be properly inferred from the importance of every single element. For instance, considering a coffee cup in the image, a classifier can still recognize the cup if a single pixel is missing. A class of methods implicitly consider the group-effect of input elements by exploiting hidden neurons, because these hidden neurons each correspond to a collection of input elements. These methods assign importance val-

ues to neurons in the last convolutional layers [23, 32] or hidden biases in all layers [27]. These methods generate attribution maps only for convolutional networks and as a result of rescaling operations, the maps are not fine-grained. Another class of methods that consider collection of input elements are perturbation mask methods [7, 6, 20, 19, 29]. These methods find the minimum set of input elements that their preservation keeps the output constant, i.e. find the maximum set of input elements that their removal does not affect the output. However, the solution to the optimization is prone to being an adversarial solution [7, 6]. Therefore certain priors such as smoothness are adopted to derive representative masks and therefore the resulting masks are limited to smooth masks [7, 6, 20].

We propose a method that inherently accounts for interactions between input elements by exploiting hidden neurons. Our method finds an input perturbation that maximally changes the output by exclusively perturbing important neurons on the path to the output. This is achieved by pruning away unimportant neurons and subsequently finding the input perturbation that maximally perturbs the output of the target neuron in the pruned network. Unimportant hidden neurons are considered to be the neurons that their removal minimally affects the output. Our method uses a first-order approximation of the effect of removing a neuron on the output. Therefore identifying the least important neurons only requires one gradient computation step. Having pruned the unimportant neurons, The remaining pruned network is solely composed of important neurons, therefore the pruned network is only sensitive to important features. In the final step, the local input perturbation that maximizes the output of the pruned model is computed. The resulting input perturbation reflects the important features and serves as a fine-grained explanation for the output of the network. We find this perturbation using two solutions. The first one is an accurate iterative solution using the projected gradient descent (PGD) algorithm. The second solution linearly approximates the pruned model, and the gradient of the output of the pruned network serves as the perturbation. We refer to the latter as PruneGrad, and demonstrate that it yields similar results as the former solution.

We emphasize on an impartial evaluation of our methods, as relying on visual evaluations and results that seem more interpretable to humans leads to confusion about whether methods indeed reflect model behavior [11, 10, 1, 17]. Therefore, we evaluate our methods against others in three acclaimed benchmarks: 1) Sanity checks [1] where the method's sensitivity to model parameter randomization is measured, 2) Pixel perturbation [22] and 3) Remove-and-Retrain (ROAR) [11, 10]. The last two evaluate whether the highlighted features by the method are in fact highly contributing features for the network. The following are the main contributions of this paper:

- We propose a novel method for providing fine-grained explanations of the output of a neural network given an input. Our method finds an input perturbation that maximally changes the output neuron by exclusively perturbing important hidden neurons (i.e. learned features) on the path to output neuron via:
    1. Pruning unimportant neurons, i.e. neurons that their removal affects the output of the target neuron the least
    2. Finding input perturbation that maximally changes the output of the target neuron in the pruned network
- We propose PruneGrad, a gradient-based and efficient solution for the pruning and perturbation steps
- Our method achieves state of the art results in three acclaimed benchmarks, 1) sanity checks [1], 2) pixel perturbation [31] and 3) Remove and Retrain (ROAR) [11, 10]

2. Related Work

2.1. Evaluation of attribution methods

Early evaluations rely on the human's perception of what is interpretable. However, there is the caveat that although the attributions seem reasonable to humans, they do not necessarily reflect model behavior. [17] Nie et al. showed theoretically and experimentally that certain attribution methods with human interpretable attributions perform partial input recovery. [1] Adebayo et al. further investigated this issue and laid out a set of sanity checks for attribution methods to pass. The sanity checks are experiments that evaluate the method's sensitivity to the model's parameter randomization and to label randomization. Several works propose evaluating the attribution methods by using theoretical axioms that are desirable for attribution methods to satisfy [28, 24, 14]. Another group of evaluation methods adopt the notion of referenced-based importance directly in their evaluation. In a pioneering work Samek et al. [22] proposed removing pixels in the image based on the scores in the attribution map, and the effect on output shows whether the computed scores are reliable. As this effect on output might be as a result of the network not having seen the perturbed input during training. Hooker et al. [11, 10] further improved on this idea by introducing Remove and Retrain (ROAR) framework, the network is retrained on the modified inputs and the drop in accuracy is regarded as the effectiveness of the attribution method.

2.2. Gradient-based attribution methods

Local importance: Simonyan et al. [25] and Baehrens et a. [5] assume a locally linear behavior of the mode and

propose the input gradient itself as a means of showing the importance of each input element

Modified backpropagation conditions: Guided Backprop [26] and RectGrad [12] set specific conditions for backpropagating gradients. Guided Backprop only allows positive gradients to be propagated at each neuron. It is shown that GBP performs partial image recovery and is invariant to sanity checks[1, 17]. Rectgrad sets a more strict condition than GBP, and only allows backpropagating gradients when the product of that gradient and its corresponding activation are larger than a threshold.

Reference-based: Another way to look at feature importance is to see the effect of not just the local change of input elements, but by changing to a reference value such as zero (i.e. removing that element). LRP [4] and DeepLift [24] methods use modified gradient propagation approaches for backpropagating the difference between output and reference output. Integrated-Gradients method [28] computes the contribution of each element, by integrating the gradients with respect to that element, while the element changes from reference to the current input.

Using high-level features: These methods leverage hidden neurons and their gradients, hence capture high-level representations. GradCAM [23] and CAM [32] perform a weighted sum of last convolutional feature maps. Fullgrad [27] incorporates high-level information by using biases and their corresponding gradients at each layer.

2.3. Perturbation-based attribution methods

Single/patch occlusion: These methods set one or multiple elements to a specific reference (baseline) value. Zeiler et al. [31] occlude a patch of pixels and observe the output change. Using a patch of pixels, as it captures the notion of multiple pixels as a feature, yields better results than single pixel occlusion [2]. Observing output change by removing one single element does not take the interdependence between elements into account. One solution for this is using Shapely values method to find the contribution of each element. Due to the complexity of finding this solution, several works have proposed approximate solutions namely SHAP [14] and DASP [3].

Mask perturbation: These methods mask the input with a certain reference value and aim at finding the smallest mask that keeps the output constant. Fong et al. [7, 6] propose finding meaningful perturbation masks, i.e. finding a mask that maximizes the output and regularizing the optimization with the size and smoothness of the mask. The smoothness prior avoids irregularly shaped masks. Qi et al. [20] improve the optimization process of finding the perturbation mask of [7] by using integrated gradients. Wagner et al. [29] set certain constraints on the optimization of [7] so that the optimization avoids adversarial perturbations. Fong et al. [6] further improve on their original proposal by changing the regularization terms in the optimization to constraints.

2.4. Identifying important hidden neurons

Oramas et al. [18] assign a relevance weight for the output of each neuron and perform a lasso regression on relevance weights and activations to regress the output. The resulting values for relevance weights signify the importance of corresponding neurons. They further use Guided-Backprop [26] to explain the selected important neurons. Wang et al. [30] assign control gates to the output of neurons, and using knowledge distillation to learn the value of these control gates such that the original output could be reconstructed. L1 regularization is imposed on control gates, therefore a sparse set of important features are found. In pruning literature, Lecun et al. [13] exploit both gradient and Hessian information as gradient alone may not be informative for saturated neurons. Most relevant to this work is the work of Molchanov et al. [16] where neurons are pruned based on the effect of their removal on the output. This effect is approximated using the first order Taylor approximation of the network.

3. Method

We study the problem of explaining the output of a neural network for a given input by attributing that output to the contribution of each input element. We provide this explanation by finding an input perturbation that maximally changes the output by exclusively perturbing important hidden neurons on the path to the output. As each of these important hidden neurons corresponds to features in the input, such perturbation reflects these main features. We achieve this objective by:

- Pruning unimportant neurons, i.e. neurons that their removal affects the output of the target neuron the least
- Finding input perturbation that maximally changes the output of the target neuron in the pruned network

We proceed by explaining each step in detail:

1) Pruning: The objective of this step is pruning neurons that their removal affects the output the least. The effect of removing a neuron is formally defined as:

$$\left| f\left(X, n_i^l = n_i^l(X)\right) - f\left(X, n_i^l = 0\right) \right| \tag{1}$$

where f is the function explaining the target neuron, n_i^l is the output of the neuron at layer l and index i and $n_i^l(X)$ signifies the value of that neuron given input X. We are interested in the magnitude of the effect of removing neurons. If we do not consider the absolute value, highly negative contributing neurons will be later pruned and this would have adverse effects on explaining the behavior of the model.

Computing the effect of removing each hidden neuron requires the computation in Eq. 1 to be done for all hidden neurons, which is computationally expensive due to excessive number of hidden neurons in the network. Therefore, similar to [16] we approximate the value of Eq. 1 using first order Taylor approximation:

$$\begin{aligned}&\left|f\left(X, n_i^l(X)\right) - f\left(X, n_i^l = 0\right)\right| = \\ &\left|n_i^l(X)) \nabla_{n_i^l} f(X, n_i^l = 0)\right|\end{aligned} \tag{2}$$

Hence, the effect of removing each neuron can be approximated with one backpropagation step using Eq. 2. Afterward, the neurons are scored based on their approximated effect and the lowest ranking neurons are pruned away according to a threshold on output change. (For implementation details please refer to section 4.1 and for the effect of threshold value on output change please refer to section 5)

2) Perturbation: The objective of this section is to find an input perturbation that maximally changes the output of the pruned network. Each hidden neuron corresponds to a group of input elements and represents a feature (pattern) in the input. In order to perturb each hidden neuron, the corresponding group of elements (feature) in the input should be perturbed, and for perturbing the target neuron, it is necessary to perturb the hidden neurons. The pruned network is solely comprised of important hidden neurons, and these important hidden neurons correspond to important input features. Therefore, in the pruned network, an input perturbation could only perturb the target neuron by perturbing important input features. Based on this intuition in order to find important features in the input, we search for an input perturbation that maximizes the target neuron's output in the pruned network:

$$\underset{\delta}{argmax} f_p\left(X + \delta\right) \tag{3}$$

where f_p is the output of the target neuron in the pruned network given input X, and δ is the perturbation and $|\delta|_2 < \Delta$, where Δ is the upper bound for perturbation.

Finding the solution to Eq. 3 is extensively investigated in adversarial attacks literature. In our experiments, we opt for Projected Gradient Descent (PGD), which is an iterative algorithm that is the strongest attack using the first-order information of the network [15]. Moreover, assuming a linear approximation of the function in Eq. 3, the gradient of the output with respect to the input ($\nabla_X f$) serves as an approximate solution. Using the input gradient as a solution is computationally more efficient, and in experiments, we show that it serves a good solution for the purpose of feature attribution. We refer to the method that chooses the input gradient as the solution as "PruneGrad".

Though the perturbation of the input is performed locally, the contributions of hidden neurons are not assigned locally and are assigned relative to a baseline of having hidden neurons removed. Therefore, the local input perturbations computed for the pruned model do not reflect the local sensitivity of the original model.

Input perturbations on the original unpruned network are liable to exhibiting adversarial effects [8]. Such perturbations, result in new critical data routing paths [30], i.e., new hidden neurons becoming important. In this scenario, input perturbations highlight other features in the input than the original highly contributing features. Restricting the model to already existing contributing neurons avoids getting adversarial effects and new evidence in the input [29]. As we have already pruned the network, and only contributing neurons remain, generating new features in the input by perturbation is strictly avoided.

4. Experiments and Results

Baseline methods: We compare our methods with GradCAM [23], recently proposed RectGrad [12], Integrated Gradients [28], Guided Backprop [26], Gradient×input [24] and pure gradient [25] (Vanilla Gradient). Shrikumar et al. [24] showed that LRP is equivalent to gradient×input (within a scaling factor), therefore we selected the later due to its simplicity. Ancona et al. [2] state that DeepLift could be deemed as a fast approximation to Integrated Gradients, therefore we compare our method with the latter to indirectly compare it with DeepLift.

4.1. Implementation details

Pruning details: As stated in section 3 we remove the neurons with the least importance scores. In this section, we clarify the procedure for removing the neurons. Our proposed general approach is to iteratively remove a certain percentage of neurons (e.g., 1%) based on their importance scores until the output changes more than the allowed threshold (the effect of the threshold is discussed in section 5). In practice, on ResNet-50 and ImageNet this approach requires ~ 10 steps for steps of 1%, hence comparably less than Integrated Gradients (note that roughly 50% of neurons are already unactivated and can be pruned in one step) However, instead of following the iterative approach, in our experiments we find the pruning threshold using a validation set. The pruning percentage(e.g., 60%) is chosen such that on average its effect on output change is equal to the allowed threshold. This trick reduces the number of required iterations of PruneGrad to two iterations, one for identifying unimportant neurons and the second for perturbation on input. In all our experiments we set the output change threshold to 15% (Results from other thresholds are provided in supplementary materials). For experiments with ResNet-50, 1000 images from the ImageNet validation set were used to specify the pruning threshold. For CIFAR10 experiments, the pruning threshold was chosen based on 10% of

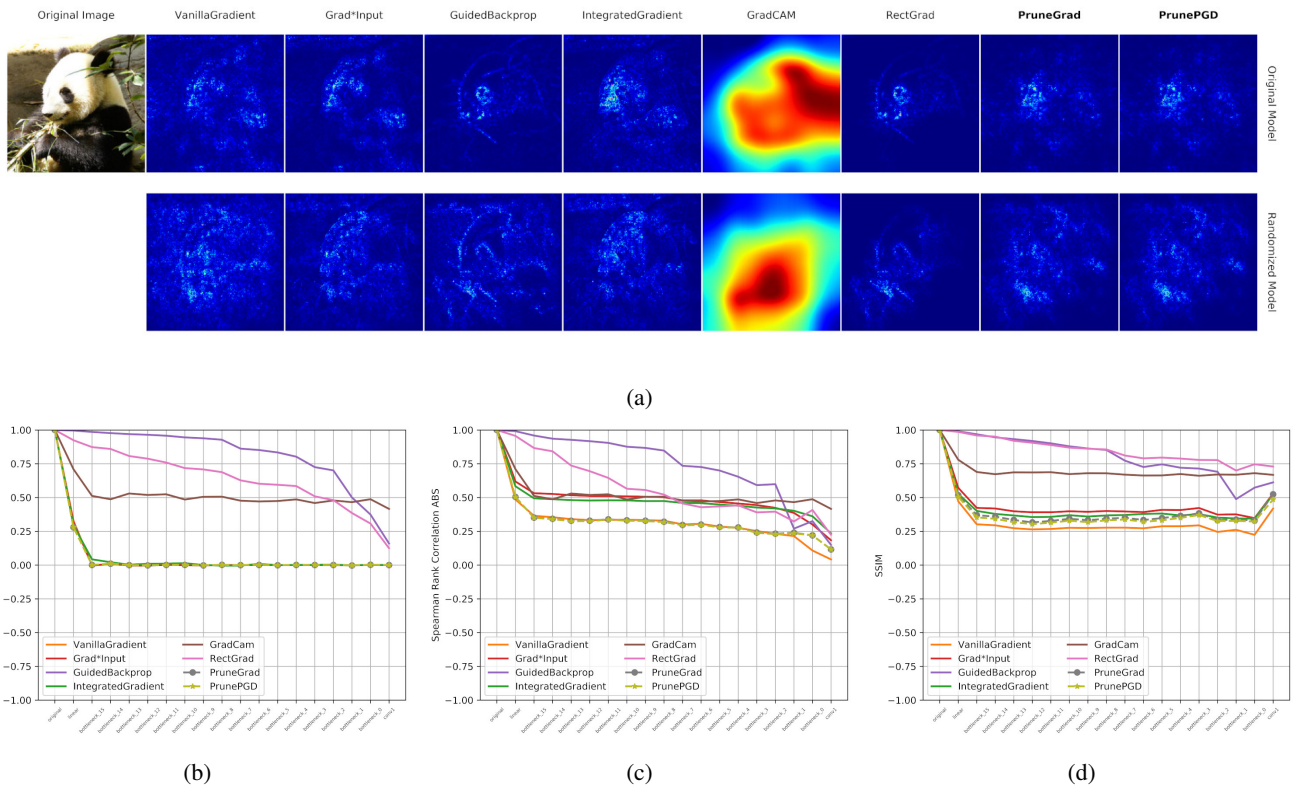


Figure 2: **Sanity Checks:** (a) The first row shows the explanations provided by various attribution methods for the prediction of a pre-trained ResNet-50 network given the Panda image. The second row shows explanations of attribution methods when all parameters in all residual blocks of the ResNet-50 network are randomized. The similarity between explanations before and after randomization implies that the explanation method is not explaining model behavior. (b), (c), (d) Three similarity metrics for comparing the original explanations and explanations after randomization (results averaged over 1k images from ImageNet test set). The x-axis shows the layers/blocks that randomization has been applied up to, while the y-axis shows Spearman rank correlation, without applying the absolute function to the explanations in (b), and with absolute values in (c). In (d), the y-axis shows SSIM. In all three metrics, the lower the curve the better.

the training set which was split as the validation set. In all of our experiments on CIFAR10, we refer to the remaining 90% as the training set.

Perturbation details: The solution to the optimization problem (Eq. 3) is once computed using Projected Gradient Descent (PGD) with the following parameters: iteration 20, a step of 0.01 and L2 bound of 0.1. This solution is represented as PrunePGD in the experiments. As explained in section 3, our efficient solution, called PruneGrad, uses the input gradient as the approximate solution of Eq. 3 in order to find the perturbation.

4.2. Sanity Checks

In this section, we conduct sanity check [1] experiments to evaluate the sensitivity of our methods to the network parameter randomization. In this experiment, all learnable parameters of the network are randomly initialized, starting from the last layer to the first layer in a cascading manner. At each randomization step, the similarity between the generated attribution map from the original network is compared with the one from the new randomized network. It is expected that attribution methods be sensitive to such randomizations, as the behavior of the network is changed via these modifications. We use a ResNet-50 [9] network that is pre-trained on ImageNet [21] and reinitialize its parameters with a normal distribution with zero mean and a standard deviation of 0.01. Fig. 2a shows the attribution maps of our proposed methods in comparison with other methods, before any reinitialization (first row), and after all residual blocks have been reinitialized (second row). It is visually evident that our methods perform well on this test as the generated attribution map for the randomized model differs from the attribution map of the original model. On the contrary, other methods including Guided Backprop, RectGrad, Integrated Gradients and Gradient$\times$Input are less sensitive to parameter randomization. Furthermore, we con-

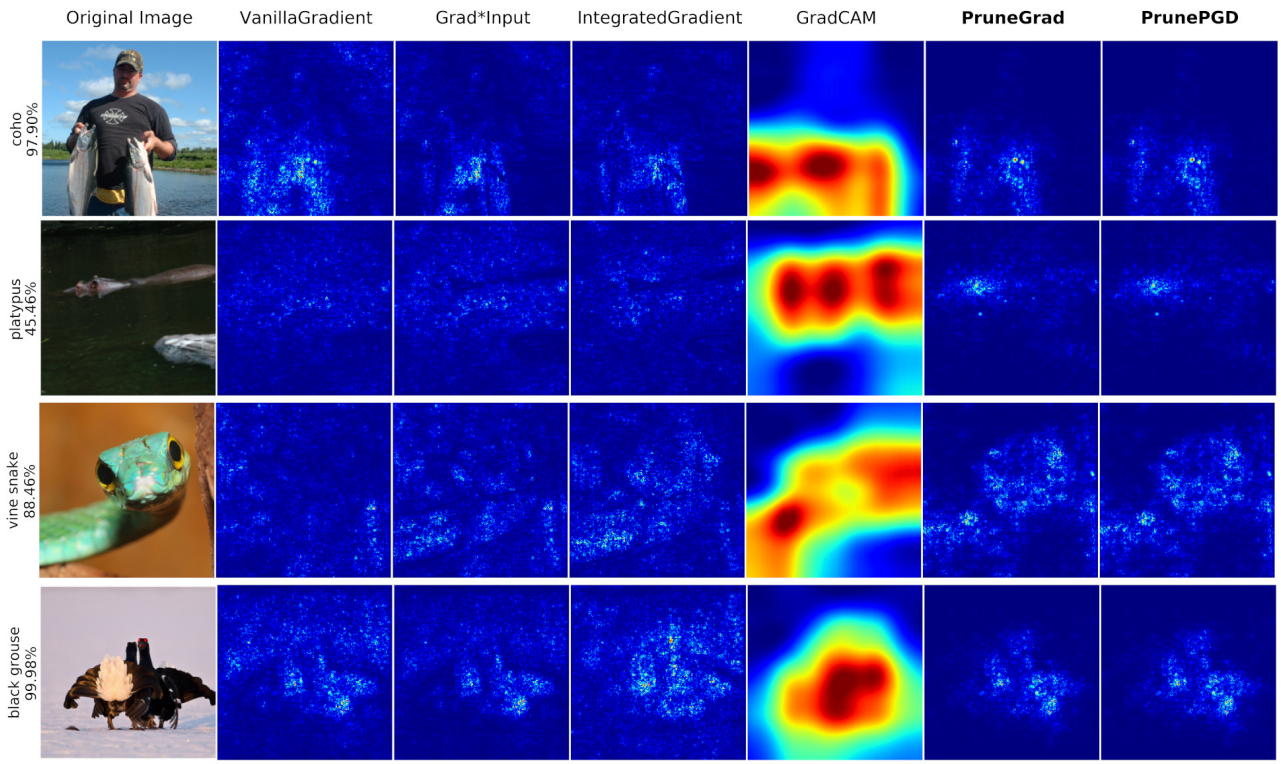


Figure 3: **Visual comparison of different attribution methods:** Our methods exclusively highlight the features relevant to the predicted output, whereas Vanilla Gradient, Gradient$\times$Input, and Integrated Gradients highlight various features in the images. GradCAM also highlights relevant features however the highlighted regions are not fine-grained, on the contrary, our proposed methods provide fine-grained explanations

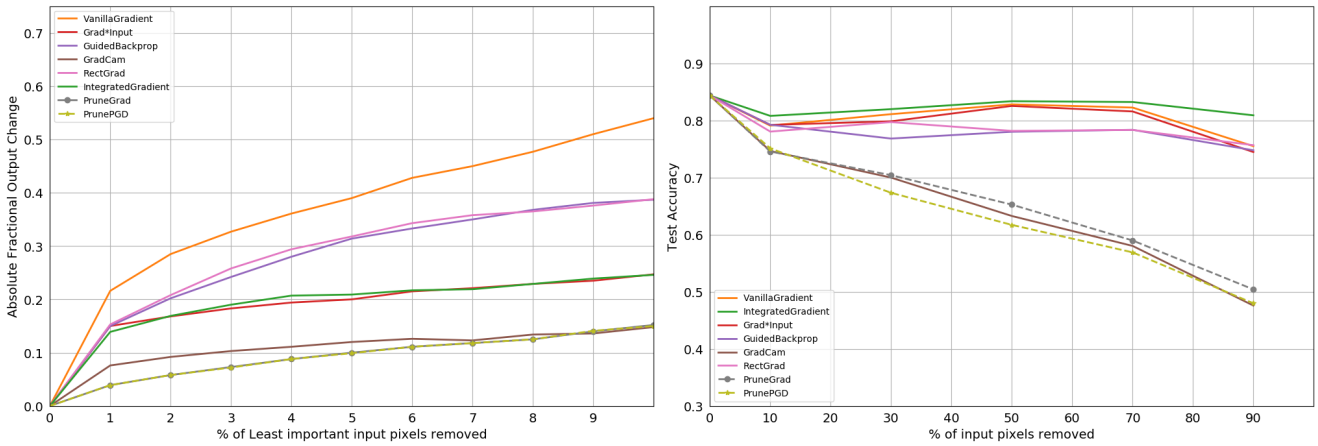
duct quantitative sanity checks. Specifically, we use the Spearman rank correlation (with and without applying an absolute function on the attribution maps) as well as the structural similarity index (SSIM) as in Fig. 2 as similarity metrics to cover different notions of similarity. The lower similarity value indicates better performance in this test. We normalize the attribution maps to range $[-1, 1]$ before calculating similarity scores in order to ignore the special characteristics of some methods as stated in [1]. A random subset of 1000 images from ImageNet [21] test set is used to evaluate attribution methods. As shown in the figure, our methods' similarity scores curve is among the bottom two (the second lowest in SSIM after VanillaGradient and the lowest in the other two). This low similarity between maps derived from the model before and after randomization shows that our methods pass these checks along VanillaGradient which reportedly [1] passes the checks.

4.3. Visual Evaluation

We conduct comparative visual experiments against baseline attribution methods that sufficiently passed sanity checks. The results are presented in Fig. 3. Input gradients (Vanilla Gradient) reveals local sensitivity information and does not demarcate features that are relevant to network's prediction. Integrated Gradients and Gradient$\times$Input tend to generate attributions similar to the input image. This may be due to the dominating input term in their mathematical formulation. Note that these methods did not score well in quantitative sanity checks. GradCAM method tends to highlight features that are relevant to the predication of the model. However, the resulting attribution maps are smooth (due to rescaling and interpolation from feature map scale). Our methods (PruneGrad and PrunePGD) highlight features that are relevant to the output of the network and the generated attributions provide fine-grained explanations.

4.4. Pixel Perturbation

This experiment evaluates attribution methods by observing the effect of removing pixels based on the scores provided by the methods and is originally proposed by Samek et al. [22]. Srinivas et al. [27] posit that removing the pixels starting from the highest scores in a descending



(a) **Pixel Perturbation** (b) **ROAR**

Figure 4: (a) The effect of removing the least important pixels in the image (as determined by the attribution methods), on the absolute output change. The lower the output change the better the attribution method is. Our proposed methods, PruneGrad and PrunePGD outperform other methods. (b) The top k = [10,30,50,70,90] percent of important pixels of each image (assigned by the attribution method) in the dataset are replaced with a constant value. The drop in accuracy of the model after being retrained on perturbed dataset signifies the effectiveness of attribution method (the lower the better).

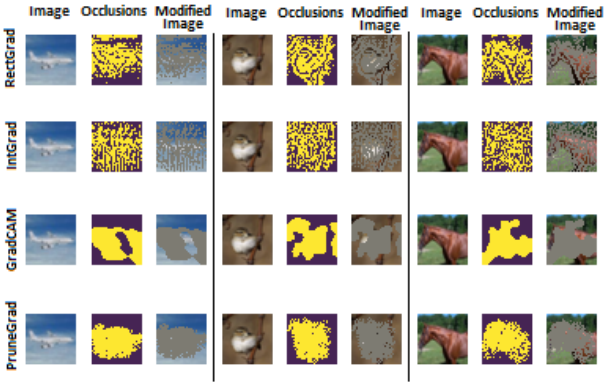


Figure 5: Three samples of the modified images in ROAR experiment where top 50% of important pixels are modified (occluded) according to the importance score assignments of different attribution methods.

order is more prone to producing artifacts for the network, therefore the output change is more likely to be a result of these artifacts than reflecting the importance of pixels. This claim is further supported in their experiments by showing that random attribution score assignment performs similar to other attribution methods if the pixels are removed in a descending order, since it creates a huge number of unnecessary artifacts that confuses the model readily. This leads to inability of distinguishing a model which provides reasonable attributions with one that creates unnecessary artifacts. Therefore in this section, we opt for removing pixels in ascending order, i.e., removing least important pixels first. We use CIFAR10 test set and a ResNet8 (three residual blocks) network trained on CIFAR10 training data. Fig. 12a shows the absolute fractional change of the output as we remove the least important pixels based on the methods explanation map. In Fig. 12a, it is clear that PruneGrad and PrunePGD outperform others in estimating unimportant pixels. This fact agrees with the pruning step of our framework in which we discard the unimportant features that contribute the least to the output.

4.5. Remove and Retrain (ROAR)

Pixel perturbation evaluation does not account for the fact that the change in output might be as a result of the network not having seen such perturbations during training. Therefore, Hooker et al. [11, 10] proposed remove-retrain (ROAR) framework to tackle this problem. Attribution maps are computed for all images in the dataset, and for each image, the top k percentage of pixels in terms of attribution scores are perturbed. The network is then retrained on the perturbed dataset. The more the resulting accuracy drops compared to the original network, the better the feature attribution method has highlighted important features. The experiment is performed on various extents of perturbation. We perform the experiments with top $k = [10, 30, 50, 70, 90]$ percentage of pixels perturbed. The experiments are carried out on CIFAR10 dataset and using a ResNet8 architecture (three residual blocks). Fig. 12b presents the resulting accuracies after the networks are

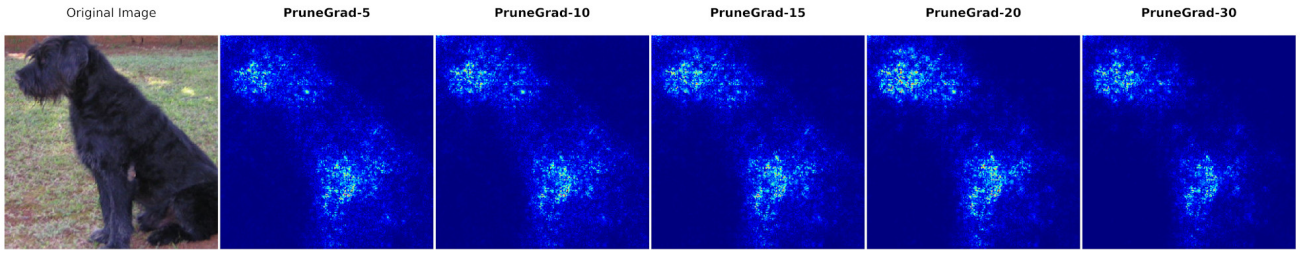


Figure 6: The effect of pruning threshold on the generated attribution maps. The extent of pruning is controlled by the output change threshold. The resulting attribution maps for different output change thresholds [5,10,15,20,30] are visualized.

trained on the modified datasets. The reported accuracies are on perturbed test sets. In order to better analyze these charts, we provide visual evidence from the modified images. In Fig. 5, we present samples of modified images where the top 50% of the pixels are removed according to the scores provided by different attribution methods. As expected, resulting perturbations of Integrated Gradients method does not conceal the main features in the image. After retraining, the model can still recognize the images. This phenomenon is also reflected in the charts in Fig. 12b as Integrated Gradients performs the worst. As the examples in Fig. 5 suggest, RectGrad mostly highlights low-level features (e.g. edges) and the corresponding perturbed images are still recognizable. As guided backprop is a special case of RectGrad [12] they are expected to highlight similar features and achieve similar results in ROAR benchmark, which is also visible in Fig. 12b. Fig. 5 shows that the modifications resulting from GradCAM and PruneGrad fully perturb the main features in these images, and this is also reflected in Fig. 12b where GradCAM, PruneGard, and PrunePGD unquestionably outperform other methods. Fig.5 shows that PruneGrad provides more fine-grained perturbations than GradCAM, however, on ROAR metric this does not seem to be of advantage, as the results in Fig. 12b for GradCAM, PruneGrad and PrunePGD are equally good.

5. Discussion

Output change threshold: As stated in section 4.1, the pruning is continued until the output changes more than a specified threshold. We have investigated the effect of setting different thresholds on the resulting attribution maps. The qualitative results are presented in Fig. 6. The figure shows that as the threshold on absolute output change increases, the attribution method focuses on more discriminative features. This results from the fact that during pruning, low contributing features are removed and as the pruning continues, only most critical features remain.

In the formulation of our proposed method, there is no assumption about the architecture of the networks or the type of activation functions. Though we did not perform experiments regarding the aforementioned points, it would be meaningful to investigate the extension to other network architectures and activations.

6. Conclusion

In this work, we proposed a novel input feature attribution method. The method finds an input perturbation that maximally changes the output neuron by exclusively perturbing important hidden neurons (i.e. learned features) on the path to output neuron. This is achieved via pruning unimportant neurons prior to finding input perturbation. The resulting perturbation serves as an explanation of important input features. We proposed PruneGrad, a gradient-based efficient solution for finding such perturbations. Our proposed solutions achieved state of the art results in three acclaimed benchmarks, namely 1) sanity checks, 2) pixel perturbation and 3) Remove and Retrain (ROAR).

References

[1] Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. Sanity checks for saliency maps. In *Advances in Neural Information Processing Systems*, pages 9505–9515, 2018.

[2] Marco Ancona, Enea Ceolini, Cengiz Öztireli, and Markus Gross. Towards better understanding of gradient-based attribution methods for deep neural networks. *arXiv preprint arXiv:1711.06104*, 2017.

[3] Marco Ancona, Cengiz Öztireli, and Markus Gross. Explaining deep neural networks with a polynomial time algorithm for shapley values approximation. *arXiv preprint arXiv:1903.10992*, 2019.

[4] 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.

[5] David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert MÃžller. How to explain individual classification decisions. *Journal of Machine Learning Research*, 11(Jun):1803–1831, 2010.

[6] Ruth Fong, Mandela Patrick, and Andrea Vedaldi. Understanding deep networks via extremal perturbations and smooth masks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2950–2958, 2019.

[7] Ruth C Fong and Andrea Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3429–3437, 2017.

[8] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[10] Sara Hooker, Dumitru Erhan, Pieter-Jan Kindermans, and Been Kim. Evaluating feature importance estimates. *arXiv preprint arXiv:1806.10758*, 2018.

[11] Sara Hooker, Dumitru Erhan, Pieter-Jan Kindermans, and Been Kim. A benchmark for interpretability methods in deep neural networks. *Advances in Neural Information Processing Systems*, 2019.

[12] Beomsu Kim, Junghoon Seo, SeungHyun Jeon, Jamyoung Koo, Jeongyeol Choe, and Taegyun Jeon. Why are saliency maps noisy? cause of and solution to noisy saliency maps. *arXiv preprint arXiv:1902.04893*, 2019.

[13] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[14] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, pages 4765–4774, 2017.

[15] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[16] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

[17] Weili Nie, Yang Zhang, and Ankit Patel. A theoretical explanation for perplexing behaviors of backpropagation-based visualizations. *arXiv preprint arXiv:1805.07039*, 2018.

[18] Jose Oramas, Kaili Wang, and Tinne Tuytelaars. Visual explanation by interpretation: Improving visual feedback capabilities of deep neural networks. *arXiv preprint arXiv:1712.06302*, 2017.

[19] Vitali Petsiuk, Abir Das, and Kate Saenko. Rise: Randomized input sampling for explanation of black-box models. *arXiv preprint arXiv:1806.07421*, 2018.

[20] Zhongang Qi, Saeed Khorram, and Fuxin Li. Visualizing deep networks by optimizing with integrated gradients. *arXiv preprint arXiv:1905.00954*, 2019.

[21] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[22] Wojciech Samek, Alexander Binder, Grégoire Montavon, Sebastian Lapuschkin, and Klaus-Robert Müller. Evaluating the visualization of what a deep neural network has learned. *IEEE transactions on neural networks and learning systems*, 28(11):2660–2673, 2016.

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

[24] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3145–3153. JMLR. org, 2017.

[25] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[26] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[27] Suraj Srinivas and Francois Fleuret. Full-gradient representation for neural network visualization. Technical report, 2019.

[28] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *Proceedings of the 34th International Conference on Machine*

Learning-Volume 70, pages 3319–3328. JMLR. org, 2017.

[29] Jorg Wagner, Jan Mathias Kohler, Tobias Gindele, Leon Hetzel, Jakob Thaddaus Wiedemer, and Sven Behnke. Interpretable and fine-grained visual explanations for convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9097–9107, 2019.

[30] Yulong Wang, Hang Su, Bo Zhang, and Xiaolin Hu. Interpret neural networks by identifying critical data routing paths. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8906–8914, 2018.

[31] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[32] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2921–2929, 2016.

7. Supplementary Figures and Charts:

- Comparative visual evaluations for ResNet50 on ImageNet: Fig. 7 and Fig. 8:
- Comparative visual evaluations for VGG16 on ImageNet: Fig. 9 and Fig. 10
- Sanity Checks (Quantitative) on PruneGrad/PrunePGD for various output change thresholds: Fig. 11
- Pixel Perturbation and ROAR evaluations on PruneGrad/PGD for various output change thresholds: Fig. 12
- More samples from CIFAR modified images dataset in ROAR experiment: Fig. 13 and Fig. 14
- Effect of output change threshold on generated attribution maps: Fig. 15

For further visual evaluations on ImageNet dataset (e.g. entire test set) please refer to the accompanying code (A Jupyter notebook is also provided for this purpose). The code also includes all experiments and will be made publicly available.

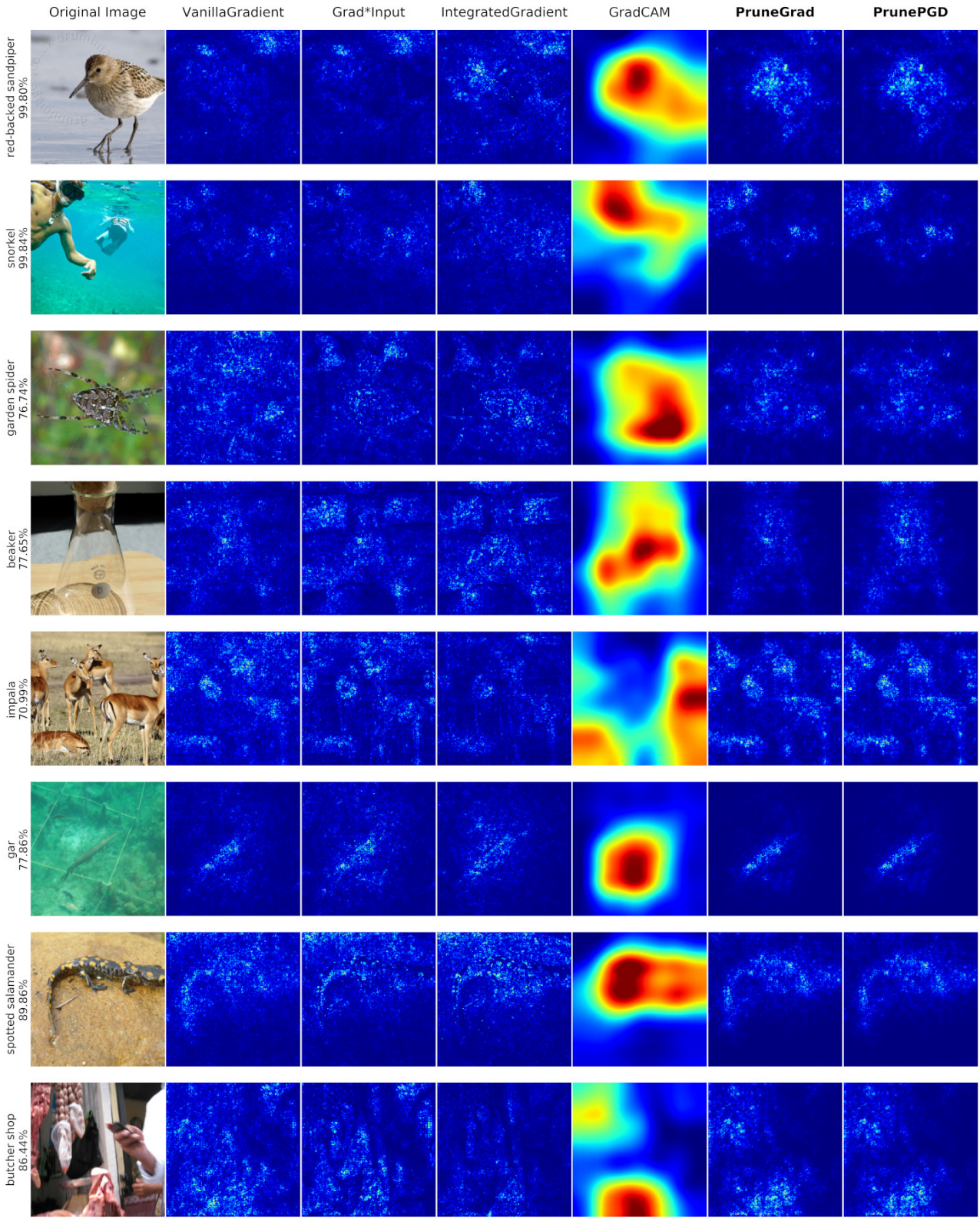


Figure 7: **Visual comparison of different attribution methods:** The attributions are computed for a pre-trained **ResNet50** network. Output change threshold for PruneGrad/PrunePGD is set to 15%.

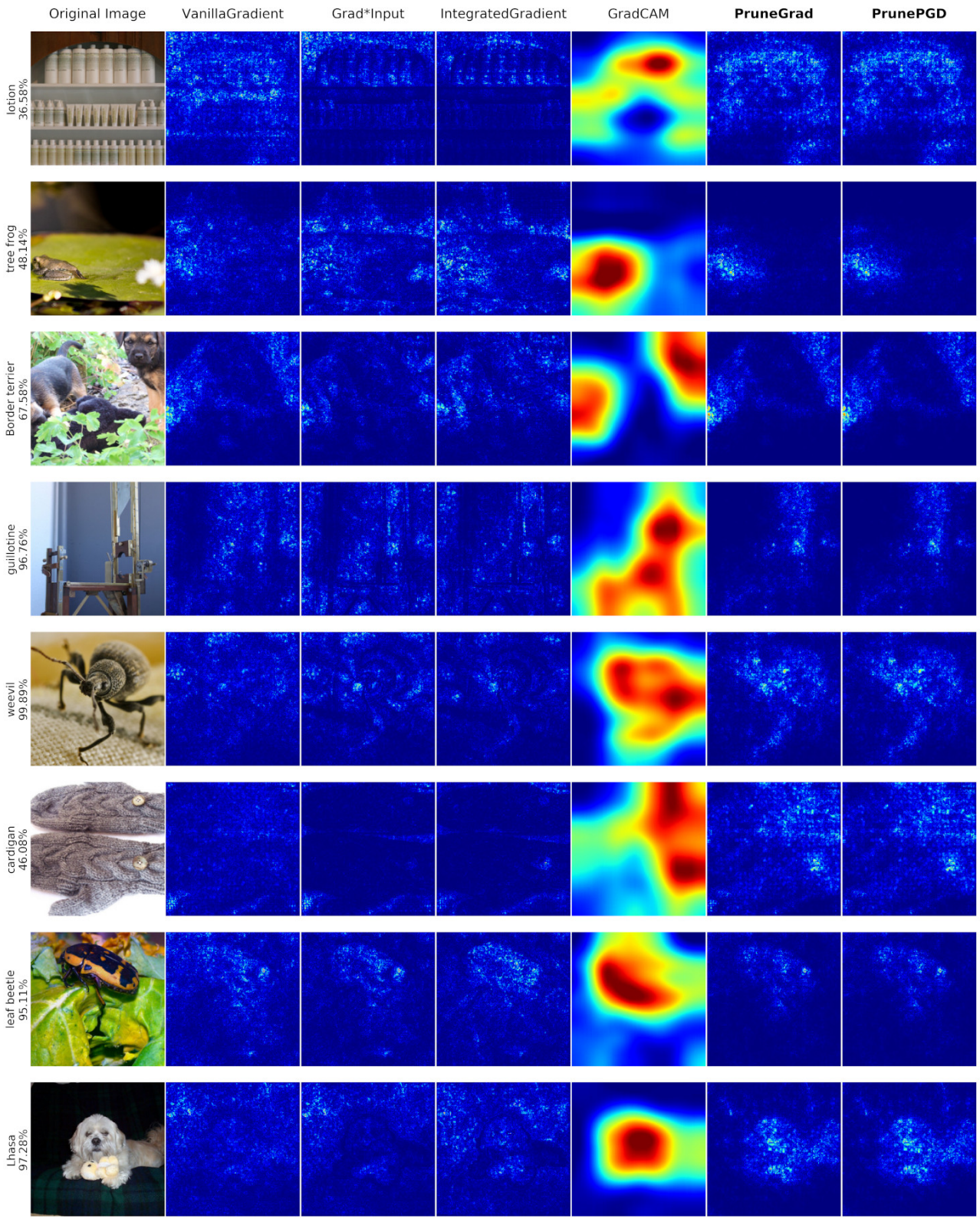


Figure 8: **Visual comparison of different attribution methods:** The attributions are computed for a pre-trained **ResNet50** network. Output change threshold for PruneGrad/PrunePGD is set to 15%.

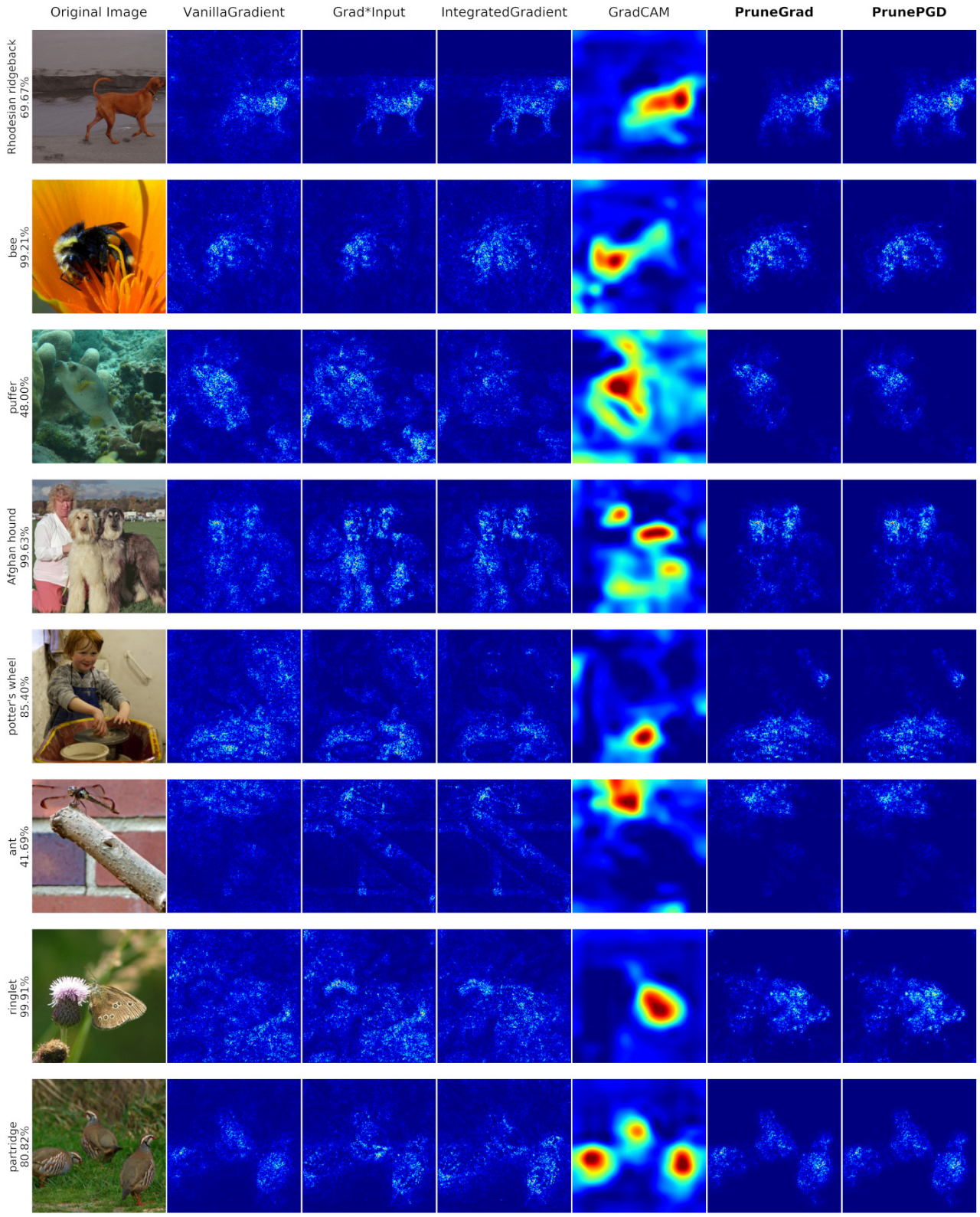


Figure 9: **Visual comparison of different attribution methods:** The attributions are computed for a pre-trained **VGG-16** network. Output change threshold for PruneGrad/PrunePGD is set to 15%.

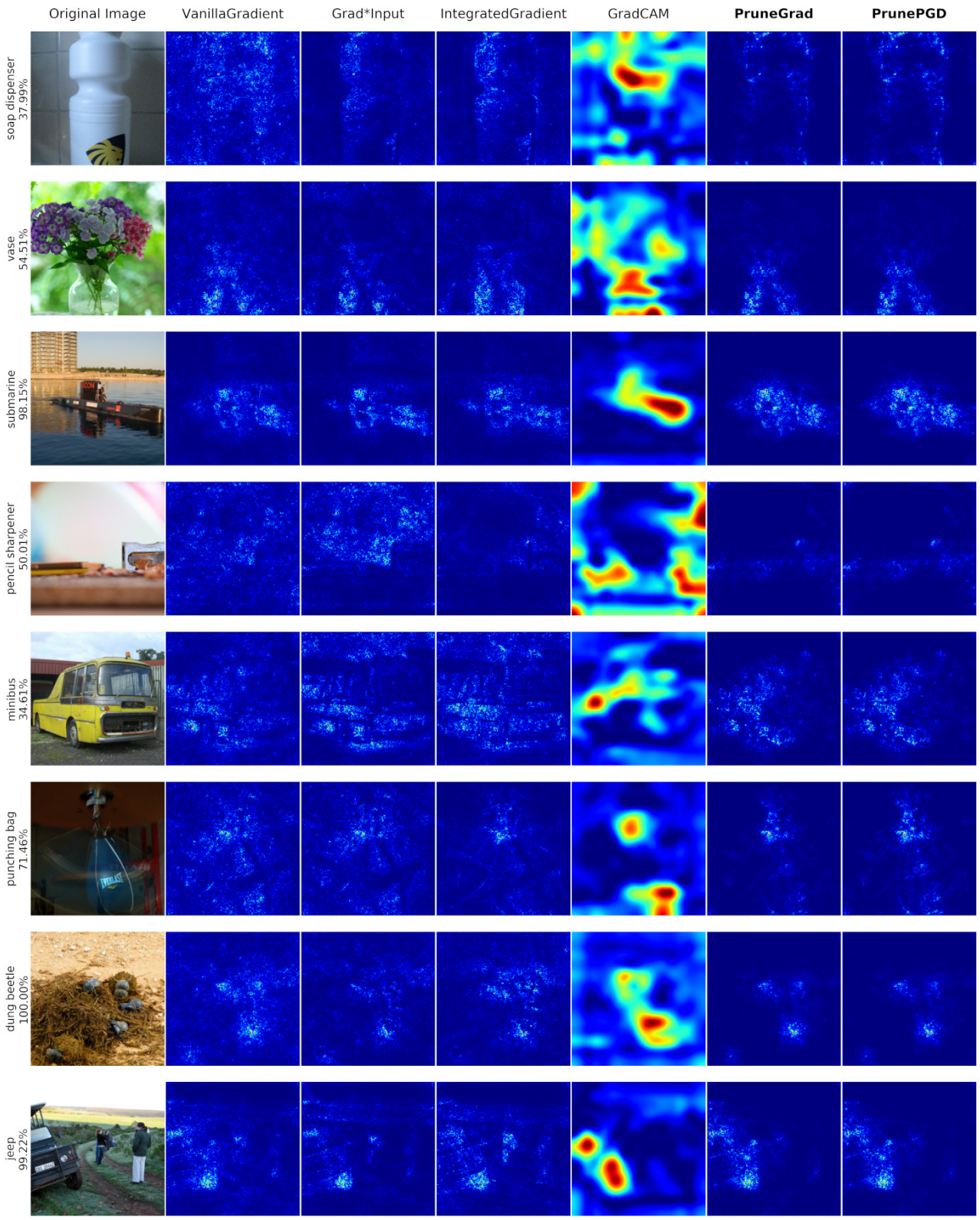
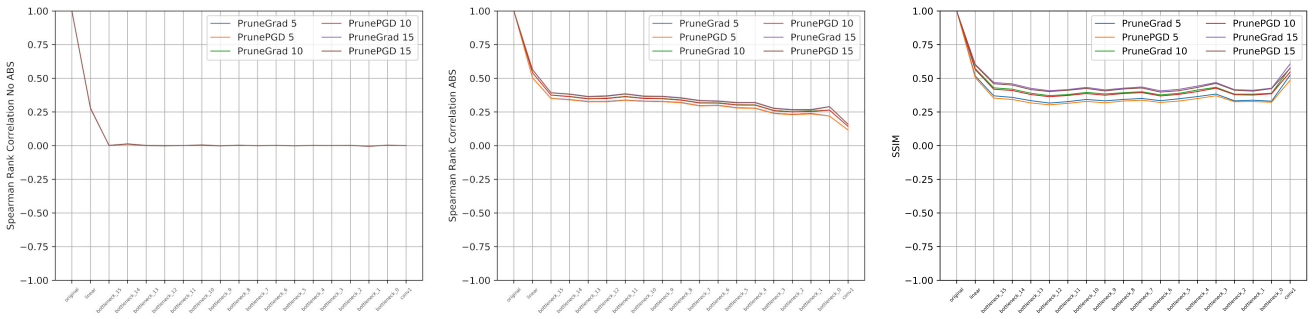
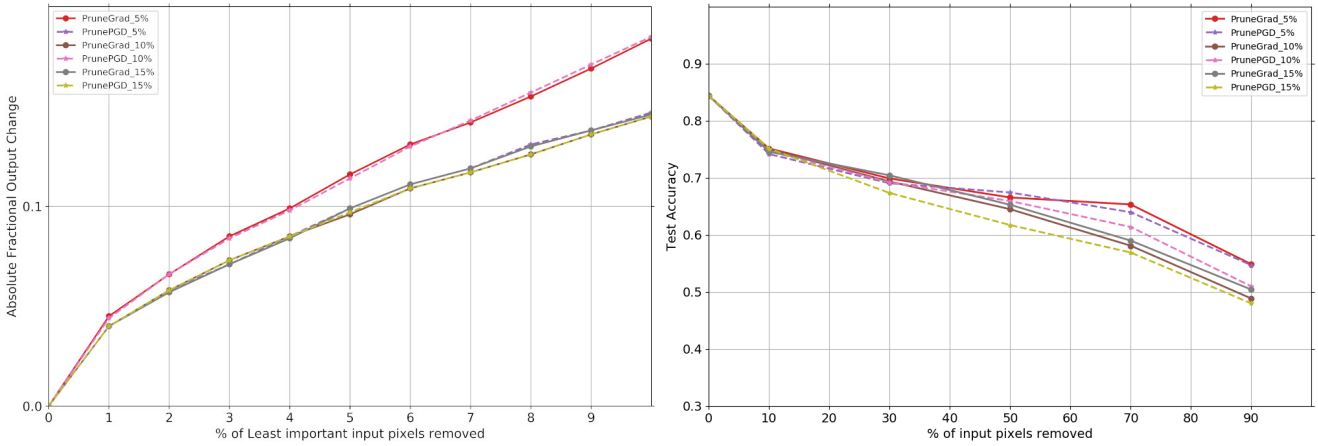


Figure 10: **Visual comparison of different attribution methods:** The attributions are computed for a pre-trained **VGG-16** network. Output change threshold for PruneGrad/PrunePGD is set to 15%.



(a) (b) (c)

Figure 11: **Sanity Checks on PruneGrad/PrunePGD for different output change thresholds:** The x-axis shows the layers/blocks that randomization has been applied up to while the y-axis shows (a) Spearman rank correlation without applying the absolute function, (b) Spearman rank correlation (absolute function applied) (c) SSIM. In all metrics results are averaged over 1k images from ImageNet test set. In all three metrics, the lower the curve the better.



(a) **Pixel Perturbation** (b) **ROAR**

Figure 12: **Pixel Perturbation and ROAR evaluations on PruneGrad and PrunePGD for different output change thresholds:** (a) The effect of removing the least important pixels in the image (as determined by the attribution methods) on the absolute output change. The lower the output change the better the attribution method is. (b) The top k = [10,30,50,70,90] percent of important pixels of each image (assigned by the attribution method) in the dataset are replaced with a constant value. The drop in accuracy of the model after being retrained on perturbed dataset signifies the effectiveness of attribution method (the lower the better).

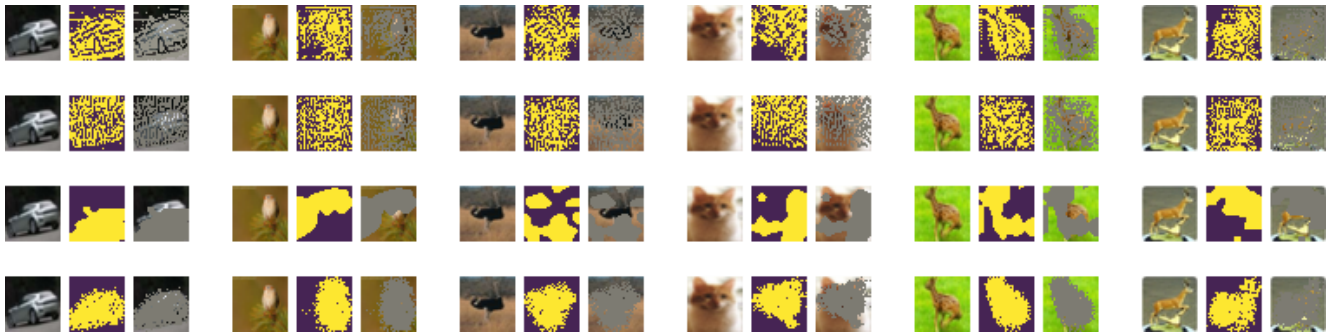


Figure 13: More samples of the modified images in ROAR experiment where top 50% of important pixels are modified (occluded) according to the importance score assignments of different attribution methods.

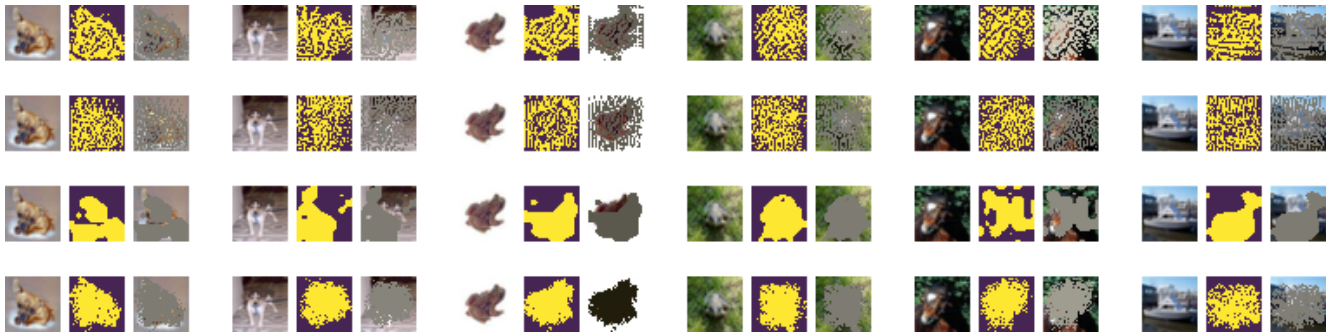


Figure 14: More samples of the modified images in ROAR experiment where top 50% of important pixels are modified (occluded) according to the importance score assignments of different attribution methods.

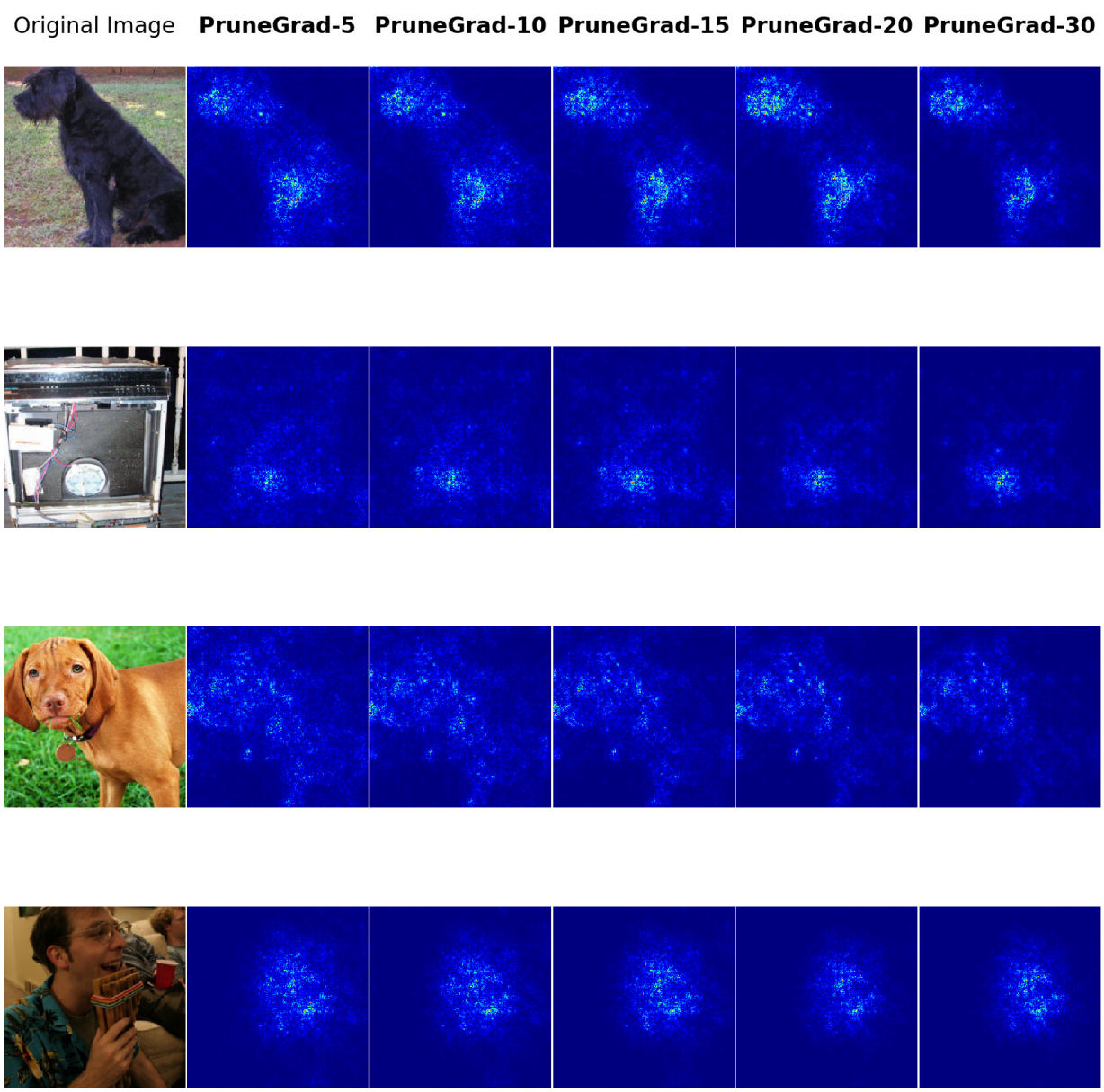


Figure 15: The effect of pruning threshold on the generated attribution maps. The extent of pruning is controlled by the output change threshold. The resulting attribution maps for different output change thresholds [5,10,15,20,30] are visualized.