GRAD-CAM++ IS EQUIVALENT TO GRAD-CAM WITH POSITIVE GRADIENTS

MIGUEL LERMA AND MIRTHA LUCAS

Abstract. The Grad-CAM algorithm provides a way to identify what parts of an image contribute most to the output of a classifier deep network. The algorithm is simple and widely used for localization of objects in an image, although some researchers have point out its limitations, and proposed various alternatives. One of them is Grad-CAM++, that according to its authors can provide better visual explanations for network predictions, and does a better job at locating objects even for occurrences of multiple object instances in a single image. Here we show that Grad-CAM++ is practically equivalent to a very simple variation of Grad-CAM in which gradients are replaced with positive gradients.

1. Introduction

Artificial Intelligence (AI) has progressed in the last few years at an accelerated rate, but many AI models behave as black boxes providing a prediction or solution to a problem without giving any information about how or why the model arrived to a given conclusion. This has a negative effect in the trust humans are willing to place in the output of AI systems. Explainable Artificial Intelligence (XAI) aims to remediate this problem by providing tools intended to explain the output of AI models. Here we look at two of them, Grad-CAM and Grad-CAM++, how they work, and how they are related.

We will we start by outlining how Grad-CAM and Grad-CAM++ work. Then, we will discuss the methodology behind Grad-CAM++, and finally we will show how Grad-CAM++ compares to a small variation of Grad-CAM that we call Grad-CAM$^+$.

2. Overview

Grad-CAM and Grad-CAM++ can be used on deep convolutional networks whose outputs are differentiable functions. Their goal is to identify what parts of the network input contribute most to the output. In this section we explain how they work.

2.1. Grad-CAM algorithm.

Here we present two versions of Grad-CAM. The first version is the algorithm as described in [3]. The second version, that we name Grad-CAM$^+$, is a small variation of Grad-CAM that we have found in some implementations of the algorithm.

2.1.1. *Grad-CAM.*

The Grad-CAM algorithm works as follows. First, we must pick a convolutional layer A, which is composed of a number of feature maps or "channels" (Figure 1). Let A^k be the k-th feature map of the picked layer, and let A^k_{ij} be the activation of the unit in position (i, j) of the k-th feature map. Then, the localization map, also called heatmap, saliency map, or attribution map, is obtained by combining the feature maps of the layer using weights w^c_k that capture the contribution of the k-th feature map to the output y^c of the network corresponding to class c.[1]

(1) Northwestern University, Evanston, USA
(2) DePaul University, Chicago, USA
E-mail addresses: mlerma@math.northwestern.edu, mlucas3@depaul.edu.

Date: May 21, 2022.

[1]We note that the Grad-CAM algorithm as described in [3] uses gradients of pre-softmax scores, we have found implementations in which gradients of post-softmax scores are used instead—which in general is easier than using pre-softmax scores e.g. when the model is given by a third party and pre-softmax scores may not be easy of access. So in general we will understand that y^c may represent either the pre-softmax or the post-softmax output of the network, at the choice of the user.

In order to compute the weights we pick a class c, and determine how much the output y^c of the network depends of each unit of the k-th feature map, as measured by the gradient $\frac{\partial y^c}{\partial A^k_{ij}}$, which can be obtained by using the backpropagation algorithm. The gradients are then averaged thorough the feature map to yield a weight w^c_k, as indicated in equation (1). Here Z is the size (number of units) of the feature map.

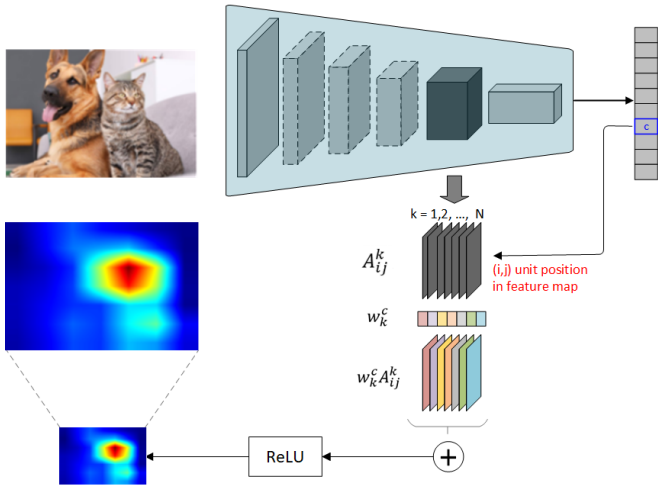

FIGURE 1. Grad-CAM overview.

$$w^c_k = \overbrace{\frac{1}{Z}\sum_i\sum_j}^{\text{global average pooling}} \underbrace{\frac{\partial y^c}{\partial A^k_{ij}}}_{\text{gradients via backprop}} . \tag{1}$$

The next step consists of combining the feature maps A^k using the weights computed above, as shown in equation (2). Note that the combination is also followed by a Rectified Linear function $\text{ReLU}(x) = \max(x, 0)$, because we are interested only in the features that have a positive influence on the class of interest. The result $L^c_{\text{Grad-CAM}}$ is called *class-discriminative localization map* by the authors. It can be interpreted as a coarse heatmap of the same size as the chosen convolutional feature map.

$$L^c = \text{ReLU} \underbrace{\left(\sum_k w^c_k A^k\right)}_{\text{linear combination}} . \tag{2}$$

After the heatmap has been produced, it can be normalized and upsampled via bilinear interpolation to the size of the original image, and overlapped with it to highlight the areas of the input image that contribute to the network output corresponding to the chosen class. Figure 2 shows the resulting heatmap (after applying a colormap) obtained in the same figure for classes 'cat' and 'dog' respectively. The red color indicates the areas in which the heatmaps have a higher intensity, which are expected to coincide with the location of the objects corresponding to the classes picked.

The method is very general, and can be applied to any (differentiable) network outputs.

2.1.2. *Grad-CAM*$^+$. In some implementations of Grad-CAM we have found that in the computation of the weights only terms with *positive* gradients are used, i.e., the weights are computed using the following

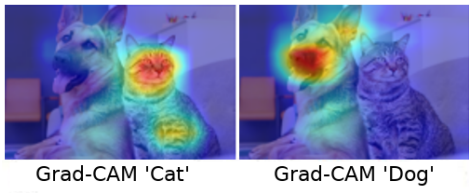

FIGURE 2. Grad-CAM locating a cat and a dog. .

formula:

$$w_k^c = \frac{1}{Z}\sum_i\sum_j \text{ReLU}\Big(\frac{\partial y^c}{\partial A_{ij}^k}\Big)\,. \tag{3}$$

This can be justified by the intuition that negative gradients correspond to units where features from a class different to the chosen class are present (say an area showing a 'cat' when trying to locate a 'dog' in an image). Although in the implementations this version of the Grad-CAM algorithm with positive gradients is still named "Grad-CAM," for clarity here we denote it "Grad-CAM$^+$."

2.2. **Grad-CAM++ algorithm.** Grad-CAM++ has been introduced with the purpose of providing a better localization than Grad-CAM [1]. When instances of an object appear in various places, or produce footprints in different feature maps, a plain average of the gradients at each feature map may not provide a good localization of the regions of interest because the units of each feature map may have different "importance" not fully captured by the gradient alone. The solution proposed in [1] is to replace the plain average of gradients at each feature map with a weighted average:

$$w_k^c = \sum_i\sum_j \alpha_{ij}^{kc}\;\; \text{ReLU}\Big(\frac{\partial y^c}{\partial A_{ij}^k}\Big)\,, \tag{4}$$

where the α_{ij}^{kc} measure the importance of each individual unit of a feature map. Note also that the gradients are fed to a ReLU function, so only positive gradients are considered at this step.

In order to determine the values of the alphas the following equation is posed:

$$Y^c = \sum_k w_k^c \;\; \sum_{i,j} A_{ij}^k\,, \tag{5}$$

i.e., the global pooling of the feature maps of the layer, combined using the weights w_k^c, should produce the class scores Y^c. Next, plugging (4) in (5) we get:

$$Y^c = \sum_k \Big(\Big\{\sum_{a,b}\alpha_{ab}^{kc}\;\; \text{ReLU}\Big(\frac{\partial Y^c}{\partial A_{ab}^k}\Big)\Big\}\Big[\sum_{i,j} A_{ij}^k\Big]\Big)\,. \tag{6}$$

In order to isolate the alphas, partial derivatives w.r.t. A_{ij}^k are computed on both sides of (6) (without the ReLU function), obtaining (equation (8) in the paper):

$$\frac{\partial Y^c}{\partial A_{jk}^k} = \sum_{a,b}\alpha_{ab}^{kc}\;\; \frac{\partial Y^c}{\partial A_{ab}^k} + \sum_{a,b} A_{ab}^k\Big\{\alpha_{ij}^{kc}\;\; \frac{\partial^2 Y^c}{(\partial A_{jk}^k)^2}\Big\}\,. \tag{7}$$

Taking partial derivative w.r.t. A_{ij}^k again and isolating α_{ij}^{kc} the following equation is obtained:

$$\alpha_{ij}^{kc} = \frac{\frac{\partial^2 Y^c}{(\partial A_{ij}^k)^2}}{2\frac{\partial^2 Y^c}{(\partial A_{ij}^k)^2} + \sum_{ab} A_{ab}^k \mathit{f}\frac{\partial^3 Y^c}{(\partial A_{ij}^k)^3}\mathit{g}}\,. \tag{8}$$

The final expression for the alphas involve second and third order partial derivatives. Assuming that the score Y^c is an exponential of the pre-softmax output of the network S^c, i.e., $Y^c = \exp(S^c)$, the expression

yielding the alphas becomes:

$$\alpha_{ij}^{kc} = \frac{\left(\frac{\partial S^c}{\partial A_{ij}^k}\right)^2}{2\left(\frac{\partial S^c}{\partial A_{ij}^k}\right)^2 + \sum_{ab} A_{ab}^k \left(\frac{\partial S^c}{\partial A_{ij}^k}\right)^3}, \tag{9}$$

which does not involve high order partial derivatives. This final equation (9) is the one used in actual implementations. Note that if $\frac{\partial S^c}{\partial A_{ij}^k} = 0$ then the right hand side of (9) becomes 0/0, which is undefined. In this case α_{ij}^{kc} is assigned value zero. After the alphas are computed the weights are obtained using equation (4), and then the salience map using equation (2).[2]

3. Discussion

In this section we discuss several issues we found in the algorithm for Grad-CAM++ and its derivation.

3.1. **The math is unclear.** In this section we focus on mathematical issues.

3.1.1. *Equation (6) is underdetermined.* Equation (6) is a system of linear equations with more unknowns (the α_{ij}^{kc}) than equations (just one per class), which makes it underdetermined. More specifically, equation (6) can be written:

$$\sum_{k,a,b} C_{ab}^k \quad \alpha_{ab}^{kc} = Y^c, \tag{10}$$

where $C_{ab}^k = \text{ReLU}\left(\frac{\partial Y^c}{\partial A_{ab}^k}\right)\left[\sum_{i,j} A_{ij}^k\right]$, or $C_{ab}^k = \frac{\partial Y^c}{\partial A_{ab}^k}\left[\sum_{i,j} A_{ij}^k\right]$ if we remove the ReLU. Note that (10) has infinitely many solutions. A general set of solutions can be written like this:

$$\alpha_{ab}^{kc} = \beta_{ab}^{kc} \quad Y^c \quad \left(\sum_{k,a,b} C_{ab}^k \quad \beta_{ab}^{kc}\right)^{1}, \tag{11}$$

where β_{ab}^{kc} are arbitrary numbers constrained only by the condition $\sum_{k,a,b} C_{ab}^k \quad \beta_{ab}^{kc} \neq 0$. Consequently, isolating the α_{ij}^{kc} to get a single solution like in equation (8) is impossible without adding additional restrictions.

3.1.2. *The alphas are treated as constants, however they may not be.* In the derivation of equation (7) the α_{ij}^{kc} are treated as constants, but that cannot be correct because they depend on the A_{ij}^k. In fact the method used to isolate the alphas is equivalent to "solving" the equation $\alpha x = x^2$ in α by differentiating on both sides w.r.t x and concluding that $\alpha = 2x$, which is obviously incorrect. The actual result of differentiating on both sides of this equation should read $\frac{d\alpha}{dx}x + \alpha = 2x$, which is correct, but does not help isolate the α, it only complicates unnecessarily the original equation.

3.1.3. *Issues with the computation of partial derivatives.* Even if we assume that the α_{ij}^{kc} are constant, taking partial derivative of (6) w.r.t. A_{ij}^k does not yield (7). The computation (assuming that the α_{ij}^{kc} are constant) is shown in the appendix—note the extra summation and the partial cross-derivatives.

3.1.4. *The alphas obtained may not satisfy their defining equation.* Another aspect that interferes with the process of solving equation (6) is that second degree derivatives kill linearities, so if after computing the alphas we replace Y^c with Y^c plus a linear function of the A_{ij}^k, i.e., $Y^c \;!\; Y^c + \sum_{i,j,k} \lambda_{ijk} A_{ij}^k + C$, where λ_{ijk} and C are constants, the process would lead to the same values for the α_{ij}^{kc}, even though they cannot be solutions to the original and new equations simultaneously.

[2]Note that it does not matter if we use $y^c = \exp(S^c)$, hence $\frac{\partial y^c}{\partial A_{ij}^k} = \exp(S^c)\frac{\partial S^c}{\partial A_{ij}^k}$, or $y^c = S^c$, $\frac{\partial y^c}{\partial A_{ij}^k} = \frac{\partial S^c}{\partial A_{ij}^k}$. The heatmaps obtained one way or the other differ by the constant factor $\exp(S^c)$, which will have no effect after min-max normalization of the heatmaps. It fact, because of the rapid grow of the exponential function, which may lead to numerical instability, the choice $y^c = S^c$ is arguably better than $y^c = \exp(S^c)$.

3.1.5. *About the assumption* $Y^c = \exp(S^c)$. The assumption that the score Y^c is an exponential of the pre-softmax output of the network S^c is based on the fact that the Y^c can be any (increasing) smooth function of S^c.[3] But if that is the case then the substitution $Y^c = \exp(\lambda S^c)$, where λ is any positive constant, would also be legitimate since $f_\lambda(x) = e^{\lambda x}$ is also smooth. However this would produce a different final formula for the alphas, namely:

$$\alpha_{ij}^{kc} = \frac{\left(\frac{\partial S^c}{\partial A_{ij}^k}\right)^2}{2\left(\frac{\partial S^c}{\partial A_{ij}^k}\right)^2 + \lambda \sum_{ab} A_{ab}^k \left(\frac{\partial S^c}{\partial A_{ij}^k}\right)^3}\,. \tag{12}$$

In principle there does not seem to be any particular reason to choose $\lambda = 1$ rather than any other value, and different values of λ will lead to different values for the alphas.

Leaving aside the question of the derivation of the formula for the alphas, we next discuss the formula actually used in the implementations to compute the alphas.

3.2. **Formula used in actual implementations.** The formula for α_{ij}^{kc} actually used in most implementations of Grad-CAM++ we have found is (9). Note that if we divide numerator and denominator by $(\partial S^c/\partial A_{ij}^k)^2$ we get

$$\alpha_{ij}^{kc} = \frac{1}{2 + \sum_{ab} A_{ab}^k \left(\frac{\partial S^c}{\partial A_{ij}^k}\right)}\,, \tag{13}$$

which is mathematically equivalent to (9), except when the gradients are zero, in which case $\alpha_{ij}^{kc} = 0$.

Although the formula does not contain second and third powers anymore (which reduces the risk of under or overflow) the expression is still numerically unstable, because nothing prevents the denominator from getting close to zero. In our experiments we have observed that this in fact happens occasionally. Also, if the second term in the denominator of (13) remains small compared to 2, the alphas will be approximately constant. Next, we discuss this two issues.

3.3. **In practice the alphas are nearly constant.** We have observed that the absolute value of the second term in the denominator of (13) is usually small compared to 2, and consequently the alphas tend to take values around 1/2. As an illustration of this point Figure 3 shows the boxplots for the distribution of non-zero values (after removal of outliers) of a_{ij}^{kc} and $\sum_{ab} A_{ab}^k \left(\frac{\partial S^c}{\partial A_{ij}^k}\right)$ in the last convolutional layer (block5_conv3) of a VGG16 [4] fed with a sample image. In this particular example most values of $\sum_{ab} A_{ab}^k \left(\frac{\partial S^c}{\partial A_{ij}^k}\right)$ lie in the interval ($0.2, 0.15$), and the resulting values for α_{ij}^{kc} range between 0.48 and 0.53 approximately.

This happened consistently for all images we tried, but in order to get a more solid result we plotted also the distribution of the alphas obtained after feeding the VGG16 with 10,000 images from ImageNetV2 MatchedFrequency [2]. The result is shown in Figure 4. For the boxplot we have considered non-zero values of α_{ij}^{kc} only, and removed outliers. The reason to consider only non-zero values for the alphas is that α_{ij}^{kc} is set to zero precisely when the gradients vanish, and in that case the values of α_{ij}^{kc} do not play any role because if $\frac{\partial S^c}{\partial A_{ij}^k} = 0$ then α_{ij}^{kc} ReLU$\left(\frac{\partial S^c}{\partial A_{ij}^k}\right) = 0$ regardless of the value of α_{ij}^{kc}. For the histogram on the right we have included again all non-zero values of alpha, but without removing outliers, centered at 0.5 and squeezed with an hyperbolic tangent, i.e., $\tanh(\alpha_{ij}^{kc}$ 0.5). The tanh-squeezing allows to display all the frequency bars within the interval [1, 1] even though the outliers may take values very far away from 0.5.

Consequently, we have α_{ij}^{kc} $\frac{1}{2}$, and

$$w_k^c \quad \frac{1}{2} \sum_i \sum_j \text{ReLU}\Big(\frac{\partial y^c}{\partial A_{ij}^k}\Big)\,, \tag{14}$$

[3] A smooth function is a function that is differentiable up to some desirable order.

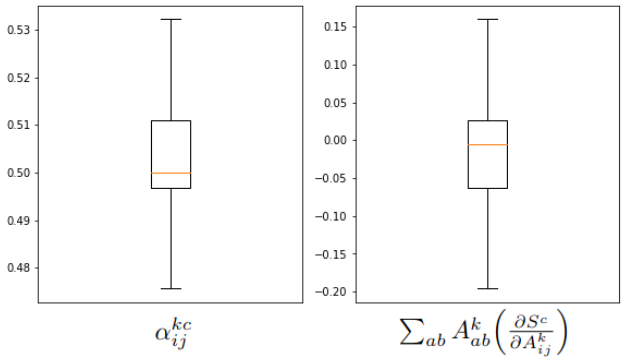

FIGURE 3. Example of distributions of non-zero α_{ij}^{kc} and $\sum_{ab} A_{ij}^{k} \left(\frac{\partial S^c}{\partial A_{ab}^{k}} \right)$ for a single image.

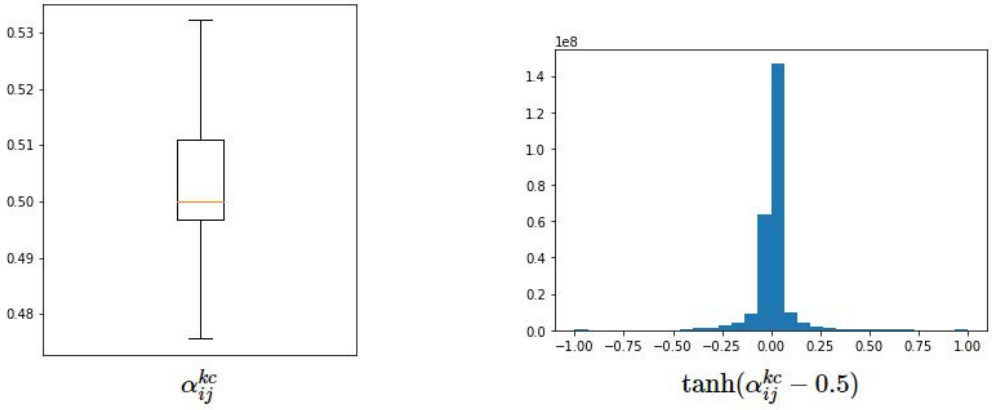

FIGURE 4. Distribution of (non-zero) α_{ij}^{kc} across 10,000 images from ImageNetV2. On the left a boxplot of the values of α_{ij}^{kc} (with outliers removed) is shown. On the right there is a histogram of the values of $\tanh(\alpha_{ij}^{kc} \quad 0.5)$.

hence, the weights computed are (except for a multiplicative constant) approximately the same as the weights computed for Grad-CAM$^+$, i.e., Grad-CAM replacing gradients with ReLU of gradients. As a consequence, in most cases we expect the heatmap produced by Grad-CAM++ to be practically the same as the one obtained using Grad-CAM$^+$ given by equation (3). Figure 5 is an illustrative example in which Grad-CAM++ and Grad-CAM with positive gradients (Grad-CAM$^+$) produce nearly identical heatmaps.

3.4. **The computation of the alphas is numerically unstable.** Even though the alphas tend to take values around 0.5 we also noticed that for a few outliers α_{ij}^{kc} reached values as large as 83.78 and 42.94 for the single image we tried. On the whole ImageNetV2 the extreme values were 1398101.4 and 559240.56 respectively, which confirmed the fact that the computation of the alphas using equation (13) is numerically unstable. Furthermore, we cannot think of any theoretical justification to assign extremely large values to the "pixel-importance" (as measured by a_{ij}^{kc}) when the value of $\sum_{ab} A_{ab}^{k} \left(\frac{\partial S^c}{\partial A_{ij}^{k}} \right)$ happens to be close to 2.

4. EMPIRICAL TESTS

The previous section contains mainly theoretical considerations, but the fact remains that the literature shows Grad-CAM++ usually performing better than Grad-CAM. Here we examine a possible explanation.

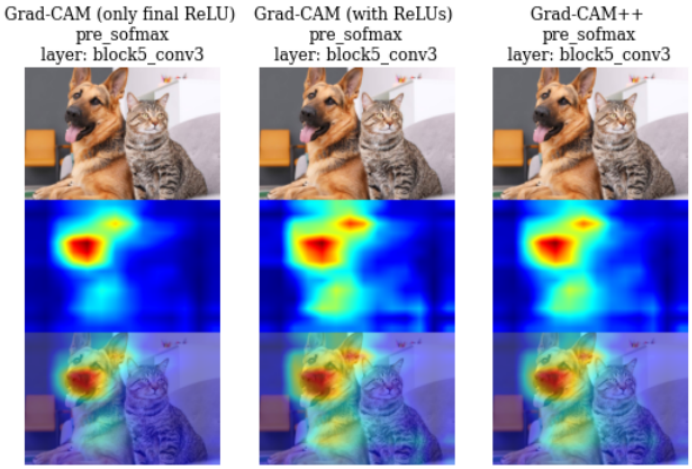

FIGURE 5. Heatmaps produced by Grad-CAM with only the final ReLU (left), Grad-CAM with positive gradients, i.e. Grad-CAM$^+$ (center), and Grad-CAM++ (right) respectively. Note that the last two heatmaps are very similar.

Our hypothesis is that the main factor that makes Grad-CAM++ better than the original version of Grad-CAM is not the Grad-CAM++ algorithm per se, but the fact that in the computation of the weights only positive gradients are used. A result that can be presented in support of this hypothesis is section 7 of [1], which shows that the performance of Grad-CAM++ drops significantly when the requirement of using only positive gradients is dropped. We will see that using only positive gradients in the computation of the weights is not only necessary, it is also sufficient in the sense that (in most cases) Grad-CAM++ does not do significantly better than Grad-CAM with positive gradients (Grad-CAM$^+$), i.e., the version of Grad-CAM in which weights are computed using equation (3). Note that the different coefficients in front of the sum are inconsequential because the heatmaps are ultimately min-max normalized.

To compare the performances of the three methods we use a metric loosely inspired in the average drop, increase in confidence, and win-percentage metrics described in [1], with a difference: rather than averaging percentages with an arithmetic mean we will average proportions using geometric mean. The reason for this choice is that, in general, adding, subtracting, and averaging percentages is not a good idea and can lead to wrong results.

The first step is to produce explanation maps, defined as the Hadamar (point-wise) product of images and their (min-max normalized) corresponding heatmaps generated by the attribution methods:

$$E^c = L^c \quad I \,. \tag{15}$$

The explanation map E^c can be interpreted as the original image I in which the areas with least contribution to the output of the network have been obscured. Hence, if we feed the explanation map E^c to the network, we expect that the output of the network will be larger when the explanation map correctly captures the relevant areas of the image compared to the output obtained if the relevant areas are poorly captured.[4]

Then, the relative performance of two attribution methods can be assessed by comparing the network outputs after feeding the network with each of the corresponding explanation maps. More specifically, let I_i be the ith image from the dataset, let E_i^c, E_i^{0c} be explanation maps produced for I_i using attribution methods M and M^0, and let O_i^c, O_i^{0c} be the network outputs (after softmax) obtained when feeding the network with E_i^c, E_i^{0c}. We call the quotient O_i^{0c}/O_i^c *relative performance* of M^0 vs M for image I_i. Note that the relative performance will be larger than 1 if the heatmap produced by method M^0 captures the relevant

[4]Note that regardless of whether explanation maps are the best way to evaluate attribution methods, they can still be used for comparison purposes since methods producing similar heatmaps will also produce similar explanation maps.

areas of I_i better than the heatmap produced by method M does, otherwise $O_i^{\emptyset c}/O_i^c$ will be less than 1. The relative performance of methods M and $M^\emptyset$ across a dataset can be measured as the geometric mean of the product of $O_i^{\emptyset c}/O_i^c$ across the given dataset:

$$\text{relative performance (across a dataset)} = \sqrt[n]{\prod_{i=1}^{n} \frac{O_i^{\emptyset c}}{O_i^c}}\,. \tag{16}$$

Note that we average the ratio $O_i^{\emptyset c}/O_i^c$ using a geometric rather than arithmetic mean. This choice is consistent with the use of the geometric mean in fields in which amounts are compared using ratios rather than differences (e.g. growth rates, financial indices, etc.). For some statistics we also use the log relative performance per image defined as $\log(O_i^{\emptyset c}/O_i^c) = \log O_i^{\emptyset c} \quad \log O_i^c$. Note that the relative performance across a dataset is the exponential of the arithmetic mean of the log relative performance across that dataset:

$$\text{relative performance} = \exp\Big\{\frac{1}{n}\sum_{i=1}^{n} \log\Big(\frac{O_i^{\emptyset c}}{O_i^c}\Big)\Big\}\,. \tag{17}$$

The testing dataset used is ImageNetV2 MatchedFrequency, which contains 10,000 images from 1,000 categories [2]. In order to separate network performance from attribution method performance, we restrict the statistics to a subset of 4219 images for which the network assigns a (post-softmax) score of more than 0.5 to the right class.

After applying the relative performance metric to each pair of algorithms of (original) Grad-CAM, Grad-CAM$^+$, and Grad-CAM++ across the dataset, we get the results shown in Table 1. The two column relative performance shows results obtained with explanation maps produced using gradients of pre-softmax and post-softmax scores respectively. As we can see, the relative performance of Grad-CAM++ and Grad-CAM$^+$ are very similar (relative performance 1), supporting our hypothesis that Grad-CAM++ is practically equivalent to Grad-CAM$^+$.

TABLE 1. Relative performances of Grad-CAM methods across the ImageNetV2 dataset for images for which the network assigns a (post-softmax) score of more than 0.5 to the right class.

methods	**relative performance**	
	pre-softmax expl. maps	post-softmax expl. maps
Grad-CAM++ vs Grad-CAM	1.24	1.27
Grad-CAM$^+$ vs Grad-CAM	1.16	1.25
Grad-CAM++ vs Grad-CAM$^+$	**1.06**	**1.01**

We get also measures of dispersion from the distributions of log relative performances per image across the dataset. We show the means and standard deviations of the distributions in Table 2.

TABLE 2. Log relative performances of Grad-CAM methods across the ImageNetV2 dataset for images for which the network assigns a (post-softmax) score of more than 0.5 to the right class.

methods	**log relative performance**			
	pre-softmax expl. maps		post-softmax expl. maps	
	mean	std	mean	std
Grad-CAM++ vs Grad-CAM	0.21	0.64	0.24	0.68
Grad-CAM$^+$ vs Grad-CAM	0.15	0.66	0.22	0.68
Grad-CAM++ vs Grad-CAM$^+$	**0.06**	**0.41**	**0.01**	**0.11**

Figure 6 shows boxplots of relative performance per image across the same dataset. A value of 1 means same performance, larger than 1 means the first method has better performance compared to the second, and less than 1 if the second does better than the first. We note that the distribution of the relative

performance again shows that Grad-CAM++ and Grad-CAM$^+$ yield very similar results, particularly if we use explanation maps obtained using gradients of post-softmax scores.

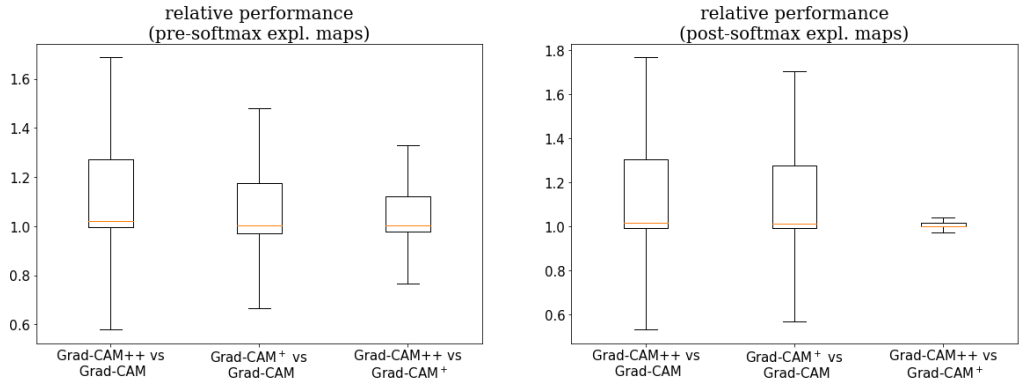

FIGURE 6. Boxplots of distributions of relative performance per image.

5. Conclusions and Future Work

We have examined the way Grad-CAM and Grad-CAM++ work, and compared them to a version of Grad-CAM, that we call Grad-CAM$^+$, in which gradients are replaced with positive gradients in the computation of the weights used to combine the feature maps that produce heatmaps. We have critically examined the methodology behind the design of the Grad-CAM++ algorithm, uncovering a number of weak points in it, and then showed that the algorithm is in fact very approximately equivalent to the much simpler Grad-CAM$^+$.

In future work additional effort can be dedicated to the comparison of Grad-CAM++ vs Grad-CAM$^+$ using different datasets and network models. Also, although the methodology used to design the Grad-CAM++ is unclear, the idea of a point-wise weighting (the alphas) during the computation of the weights (w_k^c) may still be salvaged, but equation (13) would need to be replaced with a new one that does not suffer from numerical instability and is properly justified.

References

[1] Aditya Chattopadhyay, Anirban Sarkar, Prantik Howlader, Vineeth N Balasubramanian (2018). Grad-CAM++: Improved Visual Explanations for Deep Convolutional Networks. *preprint arXiv:1710.11063 [cs.CV]*

[2] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, Vaishaal Shankar (2019). Do ImageNet Classifiers Generalize to ImageNet? *preprint arXiv:1902.10811 [cs.CV]*

[3] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, Dhruv Batra (2017). Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization. *preprint arXiv:1610.02391 [cs.CV]*

[4] Simonyan K, Zisserman A (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. *preprint arXiv:14091556 [cs.CV]*

Appendix A. Corrected Derivation for equation (7)

In equation (6) we are assuming the following:

(1) Y^c is a function of the A_{ij}^k.

(2) The A_{ij}^k are independent variables not functionally related to each other, hence $\frac{\partial A_{uv}^l}{\partial A_{ij}^k} = 1$ if $(i,j,k) = (u,v,l)$, and 0 otherwise.

(3) The α_{ij}^{kc} are treated as constants that do not depend on the A_{ij}^k, hence $\frac{\partial \alpha_{uv}^{lc}}{\partial A_{ij}^k} = 0$.

Also, following the Grad-CAM++, we remove the ReLU function.

Renaming summation indices as needed and computing the partial derivative with respect to A^k_{ij} of both sides of equation (6) (without ReLU) we get

$$
\begin{aligned}
\frac{\partial Y^c}{\partial A^k_{ij}} &= \frac{\partial}{\partial A^k_{ij}} \sum_l \Big(\Big\{ \sum_{a,b} \alpha^{lc}_{ab} \Big\} \Big[\frac{\partial Y^c}{\partial A^l_{ab}} \Big] \Big[\sum_{u,v} A^l_{uv} \Big] \Big) \\
&= \sum_l \frac{\partial}{\partial A^k_{ij}} \Big(\Big\{ \sum_{a,b} \alpha^{lc}_{ab} \Big\} \Big[\frac{\partial Y^c}{\partial A^l_{ab}} \Big] \Big[\sum_{u,v} A^l_{uv} \Big] \Big) && \text{(par. deriv. inside sum)} \\
&= \sum_l \Big(\frac{\partial}{\partial A^k_{ij}} \Big\{ \sum_{a,b} \alpha^{lc}_{ab} \Big\} \Big[\frac{\partial Y^c}{\partial A^l_{ab}} \Big] \Big) \Big[\sum_{u,v} A^l_{uv} \Big] \\
&\quad + \sum_l \Big\{ \sum_{a,b} \alpha^{lc}_{ab} \Big\} \Big[\frac{\partial Y^c}{\partial A^l_{ab}} \Big] \Big(\frac{\partial}{\partial A^k_{ij}} \Big[\sum_{u,v} A^l_{uv} \Big] \Big) && \text{(product rule)} \\
&= \sum_l \Big\{ \sum_{a,b} \alpha^{lc}_{ab} \Big\} \Big[\frac{\partial}{\partial A^k_{ij}} \Big(\frac{\partial Y^c}{\partial A^l_{ab}} \Big) \Big] \Big[\sum_{u,v} A^l_{uv} \Big] && \text{(par. deriv. inside sum)} \\
&\quad + \sum_l \Big\{ \sum_{a,b} \alpha^{lc}_{ab} \Big\} \Big[\frac{\partial Y^c}{\partial A^l_{ab}} \Big] \Big[\sum_{u,v} \underbrace{\frac{\partial A^l_{uv}}{\partial A^k_{ij}}}_{\substack{=1 \text{ if } (u,v,l)=(i,j,k) \\ =0 \text{ otherwise}}} \Big] && \text{(par. deriv. inside sum)} \\
&= \sum_l \Big(\Big\{ \sum_{a,b} \alpha^{lc}_{ab} \Big\} \frac{\partial^2 Y^c}{\partial A^k_{ij} \partial A^l_{ab}} \Big[\sum_{u,v} A^l_{uv} \Big] \Big) && \text{(combine par. deriv.)} \\
&\quad + \sum_{a,b} \alpha^{kc}_{ab} \frac{\partial Y^c}{\partial A^k_{ab}} && \text{(simplify)} \\
&= \sum_{a,b} \alpha^{kc}_{ab} \frac{\partial Y^c}{\partial A^k_{ab}} + \sum_l \Big(\Big[\sum_{u,v} A^l_{uv} \Big] \Big\{ \sum_{a,b} \alpha^{lc}_{ab} \Big\} \frac{\partial^2 Y^c}{\partial A^k_{ij} \partial A^l_{ab}} \Big) && \text{(rearrange terms)}
\end{aligned}
\tag{18}
$$

We see that the result does not mach equation (7).

If we drop the assumption that the α^{kc}_{ij} are constant, then there will be an extra term containing the partial derivatives of the alphas, and the equation becomes:

$$
\begin{aligned}
\frac{\partial Y^c}{\partial A^k_{ij}} = &\sum_{a,b} \alpha^{kc}_{ab} \frac{\partial Y^c}{\partial A^k_{ab}} + \sum_l \Big(\Big[\sum_{u,v} A^l_{uv} \Big] \Big\{ \sum_{a,b} \alpha^{lc}_{ab} \Big\} \frac{\partial^2 Y^c}{\partial A^k_{ij} \partial A^l_{ab}} \Big) \\
&+ \sum_l \Big(\Big[\sum_{u,v} A^l_{uv} \Big] \Big\{ \sum_{a,b} \frac{\partial \alpha^{lc}_{ab}}{\partial A^k_{ij}} \Big\} \frac{\partial Y^c}{\partial A^l_{ab}} \Big) .
\end{aligned}
\tag{19}
$$