

Efficient Saliency Maps for Explainable AI

T. Nathan Mundhenk¹, Barry Y. Chen¹, and Gerald Friedland^{1,2}

¹ Computational Engineering Department, Lawrence Livermore National Laboratory,
Livermore CA 94551, USA
`{mundhenk1,chen52}@lbl.gov`
<https://engineering.lbl.gov>

² Department of Electrical Engineering and Computer Sciences, University of
California at Berkeley, Berkeley CA 94720, USA
`fractor@eecs.berkeley.edu`
<https://eecs.berkeley.edu>

Abstract. We describe an explainable AI saliency map method for use with deep convolutional neural networks (CNN) that is much more efficient than popular fine-resolution gradient methods. It is also quantitatively similar or better in accuracy. Our technique works by measuring information at the end of each network scale which is then combined into a single saliency map. We describe how saliency measures can be made more efficient by exploiting Saliency Map Order Equivalence. We visualize individual scale/layer contributions by using a Layer Ordered Visualization of Information. This provides an interesting comparison of scale information contributions within the network not provided by other saliency map methods. Using our method instead of Guided Backprop, coarse-resolution class activation methods such as Grad-CAM and Grad-CAM++ seem to yield demonstrably superior results without sacrificing speed. This will make fine-resolution saliency methods feasible on resource limited platforms such as robots, cell phones, low-cost industrial devices, astronomy and satellite imagery³.

1 Introduction

Deep neural networks (DNN) have provided a new burst of research in the machine learning community. However, their complexity obfuscates the underlying processes that drive their inferences. This has lead to a new field of *explainable AI* (XAI). A variety of tools are being developed to enable researchers to peer into the inner workings of DNNs. One such tool is the XAI saliency map. It is generally used with image or video processing applications and is supposed to show what parts of an image or video frame are most important to a network’s decisions. The seemingly most popular methods derive a *gradient* saliency map by back-propagating a gradient from the end of the network and project it onto an image plane [22,30,25,26,19]. The gradient can typically be from a loss function, layer activation or class activation. Thus in order to get fine-grained details, it requires storage of the data necessary to compute a full backward pass on the input image.

³ Full source code is available at <https://github.com/LLNL/fastcam>

Several newer methods attempt to iteratively augment the image or a mask in ways that affect the precision of the results [6,1]. Additionally, saliency map encoders can be trained within the network itself [4]. Both of these methods have a distinct advantage of being more self-evidently empirical when compared with gradient techniques. *Class Activation Map* (CAM) methods [21,2,18] efficiently map a specified class to a region in an image, but the saliency map is very coarse. They generally use a method like *Guided Backprop* [25] to add finer pixel level details. This requires a full backwards pass through the network, and it adds significant memory and computational overhead to CAM solutions relative to just computing the CAM alone. Generally speaking, CAM methods compute gradients as well, but they only do so on the last few network layers. We will differentiate them from full pass gradient techniques by referring to them as CAM methods.

1.1 Efficiency and Why it Matters Here

Most fine resolution saliency map methods require many passes through the network in order to generate results or train. The gradient methods hypothetically would require just one backwards pass, but often require as many as 15 in order to give an accurate rendering [8]. This is not always a problem in the lab when a person has a powerful GPU development box. However, what if one would like to see the workings of the network at training time or even run time? This can be a serious hurdle when running a network on mobile or embedded platforms. It is not uncommon for hardware to be barely fast enough to process each image from a video source. Running a full backward pass can lead to dropped frames. Additionally, viewing or saving the saliency map for *each* frame is infeasible. Another problem is that some platforms may not have the memory capacity to save all the information required for a backward pass. Gradient based methods cannot work in these instances. Sometimes this can even be the case with powerful hardware. Satellite images can be vary large and potentially exhaust generous resources. An efficient method would enable developers in these areas to get feedback in the field and aid in debugging or understanding the behavior of a network.

Here we show a method of computing an XAI saliency map which is highly efficient. The memory and processing overhead is several orders of magnitude lower than the commonly used gradient methods. This makes it feasible in any resource limited environment. Also, since we demonstrate empirically that our method is either similar or more accurate than the most commonly used gradient methods, it can be used to speed up run-time in any situation. By combining it with an efficient CAM method, it obtains better quantitative results than state-of-the-art methods which also require quite a bit more time to compute.

2 Methods

2.1 Saliency Map Derivation

We were looking for a method to compute saliency maps based on certain conditions and assumptions.

1. The method used needs to be relatively efficient to support rapid analysis at both test time and during DNN training.
2. The method should have a reasonable information representation. As a DNN processes data, the flow of information should become localized to areas which are truly important.
3. The method should capture the intuition that the informativeness of a location is proportional to the overall activation level of all the filters as well their variance. That is, informative activations should have a sparse pattern with strong peaks.

Our approach works by creating saliency maps for the output layer of each *scale* in a neural network and then combines them. We can understand scale by noting that the most commonly used image processing DNNs work on images with filter groups at the same scale which down-sample the image and pass it to the group of filters at the next scale, and so on. Given a network like ResNet-50 [7] with an input image size of 224×224 , we would have five scale groups of size 112×112 , 56×56 , 28×28 , 14×14 and 7×7 . It is at the end of these scale groups where we are interested in computing saliency. In this way, our approach is efficient and is computed during the standard forward pass through the network. No extra pass is needed.

To achieve localization of information, we measure statistics of activation values arising at different input locations. Given an output activation tensor $\mathbf{T} \in \mathbb{R}^{+, p \times q \times r}$ with spatial indices $i, j \in p, q$ and depth index $k \in r$ from some layer $\mathbf{T} = l(\mathbf{X})$. In our case $l(\cdot)$ is a *ReLU* [17]. We apply a function to each column at i, j over all depths k . This yields a 2D saliency map $\mathbf{S} \in \mathbb{R}^{+, p \times q}$ where $\mathbf{S} = \varphi(\mathbf{T})$. We process the tensor after it has been batch-normalized [10] and processed by the activation function. When we compute Truncated Normal statistics as an alternative in later section, we take the tensor prior to the activation function.

Finally, to capture our intuition about the informativeness of an output activation tensor, we derived $\varphi(\cdot)$ by creating a special simplification of the maximum likelihood estimation *Gamma Scale* parameter [3]. One way we can express it is:

$$\varphi(\cdot) = \frac{1}{r} \cdot \sum_{k=1}^r x_k \cdot \left[\log_2 \left(\frac{1}{r} \cdot \sum_{k=1}^r x_k \right) - \frac{1}{r} \cdot \sum_{k=1}^r \log_2(x_k) \right] \quad (1)$$

To avoid log zero, we add $1e-06$ to each x_k . How mean and variance relate seems readily apparent with the square bracketed part being the computational formula for the standard deviation with values taken to $\log_2(\cdot)$ rather than squared. This is preceded by a typical mean estimate. This meets the third requirement we mentioned. This simplification is **Saliency Map Order Equivalent (SMOE)** to the full iterative (and expensive) scale parameter estimation. We define SMOE as follows. Given saliency map $\mathbf{S}_a \in \mathbb{R}^{+, p \times q}$ and $\mathbf{S}_b \in \mathbb{R}^{+, p \times q}$ where we may have $\mathbf{S}_a \neq \mathbf{S}_b$, if we sort the pixels by value, then \mathbf{S}_a will be sorted in exactly the same order as \mathbf{S}_b . That means that the most salient location i, j is exactly the same in both \mathbf{S}_a and \mathbf{S}_b . This also means that if we create a binary mask of the $n\%$ most salient pixels, the mask for both \mathbf{S}_a and \mathbf{S}_b will also be exactly

the same. SMOE is preserved if for instance, we apply independent monotonic functions to a saliency map. As such, we may as well strip these away to save on computation. Tie values may create an imperfect equivalence, but we assert that these should be very rare and not affect results by a measurable amount.

Using μ as the mean of each column r in \mathbf{T} , we can see the information relation more clearly if we simplify Eq 1 further which gives us our **SMOE Scale** method:

$$\varphi(.) = \frac{1}{r} \cdot \sum_{k=1}^r \mu \cdot \log_2 \left(\frac{\mu}{x_k} \right) \quad (2)$$

The resemblance to conditional entropy is apparent. However, since the values in Eq 2 are not probabilities, this does not fit the precise definition of it. On the other hand, the interpretation is fairly similar. It is the mean activation multiplied by the information we would gain if we knew the individual values which formed the mean. Put in traditional terms, it is the information in the mean conditioned on the individual values. The method is sensitive to variations beyond the mean value which might be uninformative if all activations are similar. Since activations are principally the result of products, using a log statistic may also make more sense. Numerical examples of this method at work can be seen in *Appendix* Table 3 along with more information on the derivation. To create a 2D saliency map $\mathbf{S} \in \mathbb{R}^{+, p \times q}$, we simply apply Eq 2 at each spatial location $i, j \in p, q$ with column elements $k \in r$ in the 3D activation tensor \mathbf{T} for a given input image.

2.2 Combined Saliency Map Generation

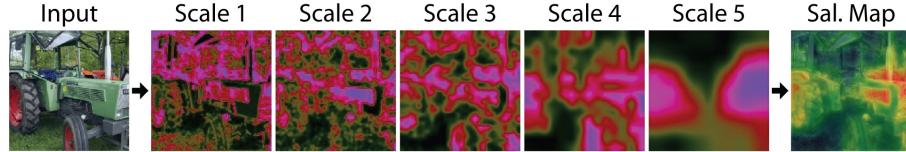


Fig. 1. The left most image is the input to the network. Five saliency maps are shown for each spatial scale in the network. They are combined per Eq 3. The right most image is the combined saliency map created from these. To aid in visualizing context, it has been alpha blended with a gray scale version of the original image here at 25%. Many more combined saliency map examples can be seen in *Appendix* Figures 10 and 11

For each input image, we derive five saliency maps. For different networks, this number may vary. Given a network such as a ResNet [7], AlexNet [15], VGG Net [23] or DenseNet [9] we compute saliency on the final tensor computed at each spatial scale. Recall that most image classification networks process images

in a pipeline that processes an image in consecutive groups of convolution layers where each group downsamples the image by $1/2x$ before passing it onto the next. It is just prior to the downsampling that we compute each saliency map. Computing saliency across image scales is a classical technique [12]. This is also similar to the method used in the XAI saliency technique described in [4].

To make our maps easier to visualize or combine together, we normalize them from 0 to 1 by squashing them with the normal *cumulative* distribution function $\gamma(s; \mu, \sigma)$. Here mean and standard deviation are computed independently over each saliency map. We then create a combined saliency map by taking the weighted average of the maps. Since they are at different scales, they are upsampled via bilinear interpolation to match the dimensions of the input image. Given r saliency maps that have been bilinear interpolated (upsampled) to the original input image size p, q , they are then combined as:

$$c_{i,j} = \frac{\sum_{k=1}^r \gamma(s_{i,j,k}; \mu_k, \sigma_k) \cdot w_k}{\sum_{k=1}^r w_k} \quad (3)$$

Note that technically, we compute $\gamma(s; \mu, \sigma)$ *before* we upsample. Weighting is very useful since we expect that saliency maps computed later in the network should be more accurate than saliency maps computed earlier as the network has reduced more irrelevant information in deeper layers, distilling relevant pixels from noise [27]. Network activity should be more focused on relevant locations as information becomes more related to the message. We observe this behavior which can be seen later in Figure 4. A saliency map generation example can be seen in Figure 1 with many more examples in *Appendix* Figures 10 and 11.

The advantages of creating saliency maps this way when compared with most gradient methods are:

- **Pro:** This is relatively efficient, requiring processing of just five low cost layers during a standard forward pass.
- **Pro:** We can easily visualize the network at different stages (layers).
- **Con:** It does not have a class specific activation map (CAM) [21,2,18], but we show how this can be integrated later.

2.3 Visualizing Multiple Saliency Maps

One advantage to computing multiple saliency maps at each scale is that we can get an idea of what is happening in the middle of the network. However, with so many saliency maps, we are starting to be overloaded with information. This could get even worse if we decided to insert saliency maps after each layer rather than just at the end of each scale. One way to deal with this is to come up with a method of combining saliency maps into a single image that preserves useful information about each map. Such a composite saliency map should communicate where the network is most active as well as which layers specifically are involved. We call our method **Layer Ordered Visualization of Information (LOVI)**. We do this by combining saliency maps using an *HSV* color scheme [14] where *hue* corresponds to which layer is most active at a given location. That is, it shows the mean layer around which a pixels activity is centered. *Saturation* tells

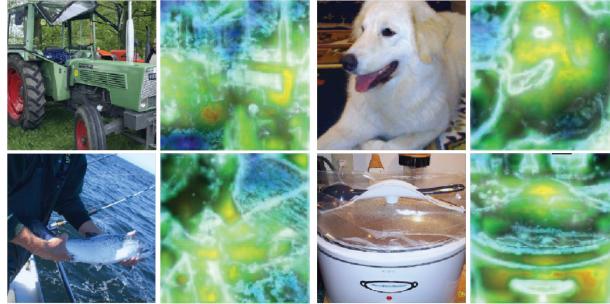


Fig. 2. Images are shown with their combined saliency map using our *LOVI* scheme. The hue in each saliency map corresponds to layer activation. Earlier layers start at violet and trend red in the last layers following the order of the rainbow. Areas which are blue or violet are only activated early in network processing. They tend to activate early filters, but are later disregarded by the network. Yellow and red areas are only activated later in the network. They appear to be places where the objects components are combined together. White areas are activated throughout all network layers. They possibly correspond to the most important features in the image. Many more examples can be seen in *Appendix* Figures 12 and 13

us the uniqueness of the activation. This is the difference between the maximum value at a location and the others. *Value* (intensity) corresponds to the maximal activation at that location. Basically, this is a pixel's importance.

If only one layer is active at a location, the color will be very saturated (vivid colors). On the other hand, if all layers are equally active at a given location, the pixel value will be unsaturated (white or gray). If most of the activation is towards the start of a network, a pixel will be violet or blue. If the activation mostly happens at the end, a pixel will be yellow or red. Green indicates a middle layer. Thus, the color ordering by layer follows the standard order of the rainbow. Examples can be seen in Figure 2. Given $k \in r$ saliency maps \mathbf{S} (in this instance, we have $r = 5$ maps), we stack them into a tensor $\mathbf{S} \in \mathbb{R}^{+, p \times q \times r}$. Note that all $s \in [0, 1]$ because of Eq 3 and they have been upsampled via bilinear interpolation to match the original input image size. Given:

$$\phi(k) = 1 - \frac{k-1}{r} \cdot \left(\frac{r-1}{r}\right)^{-1}, \nu = \frac{1}{r} \quad (4)$$

Hue $\in [0, 360]$ is basically the center of mass of activation for column vector \mathbf{s} at each location $i, j \in p, q$ in \mathbf{S} :

$$Hue = \frac{\sum_{k=1}^r s_k \cdot \phi(k)}{\sum_{k=1}^r s_k} \cdot 300 \quad (5)$$

Saturation $\in [0, 1]$ is the inverse of the ratio of the values in \mathbf{s} compared to if they are all equal to the maximum value. So for instance, if one value is large and all the other values are small, saturation is high. On the other hand, if all values

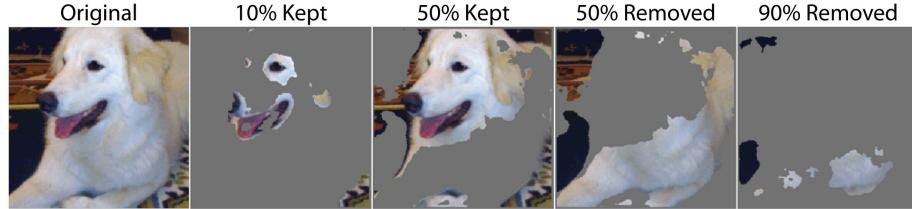


Fig. 3. These are the KAR (kept) and ROAR (removed) mask images created by masking out the original images with the combined saliency map. The percentage is how much of the image has been kept or removed based on the combined saliency map. Thus, the 10% kept example shows the top 10% most salient pixels in the image. It is these example images that are fed into the network when we compute the KAR and ROAR scores. Many more examples can be seen in *Appendix Figure 9*

are about the same (equal to the maximum value), saturation is very small:

$$Sat = 1 - \frac{\sum_{k=1}^r (s_k) - \nu}{r \cdot max(\mathbf{s}) \cdot (1 - \nu)} \quad (6)$$

Value $\in [0, 1]$ is basically just the maximum value in vector \mathbf{s} :

$$Val = max(\mathbf{s}) \quad (7)$$

Once we have the HSV values for each location, we then convert the image to RGB color space in the usual manner.

2.4 Quantification Via ROAR and KAR

[8] proposed a standardized method for comparing XAI saliency maps. This extends on ideas proposed by [4,20] and in general hearkens back to methods used to compare computational saliency maps to psychophysical observations [11]. The general idea is that if a saliency map is an accurate representation of what is important in an image, then if we block out salient regions, network performance should degrade. Conversely, if we block out non-salient regions, we should see little degradation in performance. The ROAR/KAR metrics measure these degradations explicitly. The KAR metric (Keep And Retrain) works by blanking out the *least salient* information/pixels in an input image, and the ROAR (Remove And Retrain) metric uses the contrary strategy and removes the *most salient* pixels. Figure 3 shows an example of ROAR and KAR masked image. A key component to the ROAR/KAR method is that the network needs to be retrained with saliency masking in place. This is because when we mask out regions in an input image, we unavoidably create artifacts. By retraining the network on masked images, the network learns to ignore the new artifacts and focus on image information.

We will give a few examples to show why we need both metrics. If a saliency map is good at deciding which parts of an image are least informative but gets the

ranking of the most salient objects wrong, ROAR scores will suggest the method is very good. This is because it masks out the most salient locations in one large grouping. However, ROAR will be unable to diagnose that the saliency map has erroneously ranked the most informative locations until we have removed 50% or more of the salient pixels. As such, we get no differentiation between the top 1% and the top 10% most salient pixels. On the other hand, KAR directly measures how well the saliency map has ranked the most informative locations. By using both metrics, we can quantify the goodness of both the most and least salient locations in a map.

3 Quantitative Experiments

3.1 Comparing Different Efficient Statistics

We tested our SMOE Scale saliency map method against several other statistical measures using three different datasets that have fairly different tasks and can be effectively trained from scratch. The sets used are ImageNet [5], CSAIL Places [31] and COWC [16]. ImageNet as a task focuses on foreground identification of objects in standard photographic images. Places has more emphasis on background objects, so we would expect more spatial distribution of useful information. COWC, *Cars Overhead With Context* is an overhead dataset for counting as many as 64 cars per image. We might expect information to be spatially and discretely localized, but distributed over many locations. In summary, these three datasets are expected to have fairly different distributions of important information within each image. This should give us more insight into performance than if we used several task-similar datasets (e.g. Three photographic foreground object sets such as; ImageNet + CUB (birds) [28] + CompCars [29]).

For compatibility with [8], we used a ResNet-50 network [7]. We also show performance on a per layer basis in order to understand the accuracy at different levels of the network. For comparison with our SMOE Scale method, we included any statistical measure which had at least a modicum of justification and was within the realm of the efficiency we were aiming for. These included parameter and entropy estimations from *Normal*, *Truncated-normal*, *Log-normal* and *Gamma Distribution* models. We also tested *Shanon Entropy* and *Renyi Entropy*. To save compute time, we did a preliminary test on each method and did not continue using it if the results qualitatively appeared very poor and highly unlikely to yield good quantitative results. Normal entropy was excluded because it is SMOE with the Normal standard deviation. This left us with nine possible statistical models which we will discuss in further detail.

Saliency maps for each method are computed over each tensor column in the same way as we did with our SMOE Scale metric. The only difference is with the truncated-normal statistic which computes parameters prior to the ReLU layer. We should note that [13] uses a truncated normal distribution to measure network information for network design. Recall that we have five saliency map layers. They are at the end of each of the five network spatial scales. We test each one at a time. This is done by setting the network with pre-trained weights for the specific dataset. Then, all weights in the network which come after the saliency

Table 1. KAR and ROAR results per dataset. The *Difference Score* shows the results using Eq 8. The *Information Score* uses Eq 9. They are sorted by the average difference score (AVG). The SMOE Scale from Eq 2 performs best overall using both scoring methods. The vanilla standard deviation is second best. Recall it is SMOE with normal entropy. Truncated normal entropy is best on the COWC set and ranks third overall. It is interesting to note that the difference in scores over COWC are not as large as the other two datasets. The top four methods all are information related and mean activation style methods are towards the bottom. Finer grain details can be seen in *Appendix* Table 4. Quartile rankings for methods can be seen in *Appendix* Table 7

Method	Difference Score			Information Score				
	ImNet	Places	COWC	AVG	ImNet	Places	COWC	AVG
SMOE Scale	1.70	0.90	1.61	1.40	1.13	0.68	1.31	1.04
Standard Dev	1.64	0.83	1.61	1.36	1.07	0.61	1.30	0.99
Trunc Normal Ent	1.56	0.77	1.64	1.32	1.00	0.56	1.32	0.96
Shanon Ent	1.61	0.80	1.51	1.31	0.98	0.59	1.23	0.93
Trunc Normal Std	1.51	0.71	1.64	1.28	1.00	0.52	1.32	0.94
Trunc Normal Mean	1.38	0.67	1.64	1.23	0.96	0.49	1.32	0.92
Normal Mean	1.29	0.63	1.42	1.11	0.75	0.44	1.18	0.79
Log Normal Ent	1.16	0.66	1.44	1.09	0.82	0.47	1.20	0.83
Log Normal Mean	1.46	0.55	1.09	1.03	0.54	0.35	0.88	0.59

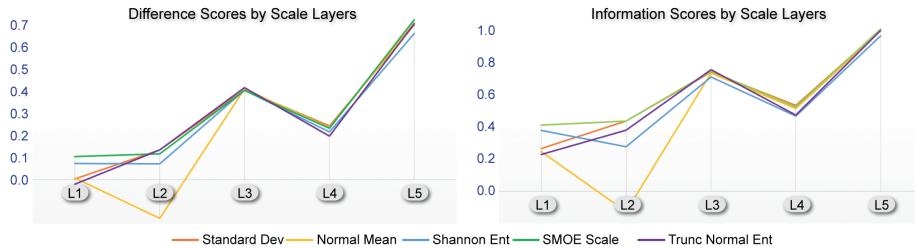


Fig. 4. SMOE Scale is compared with several other efficient statistical methods. The **Y-axis** is the **combined score per scale layer** over all three image sets. The **X-axis** is the **network layer** with L1 being the earliest layer in the network and L5 near the end. A difference score of zero means the result was about the same as for a randomly generated saliency map. Less than zero means it is worse. SMOE Scale differentiates itself the most early on in the network where most statistical methods score at the level of a random saliency map. About mid way through, the difference between methods becomes relatively small. This may be because information contains more message and less noise by this point in processing. Finer grain details can be seen in *Appendix* Table 4

mask to be tested are allowed to fine-tune over 25 epochs. Otherwise, we used the same methodology as [7] for data augmentation etc. This is necessary in order to adapt the network to mask related artifacts as specified in the ROAR/KAR protocol. At the level where the saliency map is generated, we mask out pixels in the activation tensor by setting them to zero. For this experiment, we computed the ROAR statistic for the 10% least salient pixels. For KAR, we computed a map to only let through the top 2.5% most salient pixels. This creates a more similar magnitude between ROAR and KAR measures. Layer scores for the top five methods can be seen in Figure 4.

We combine layer scores two different ways since ROAR and KAR scores are not quite proportional. These methods both yield very similar results. The *first method* takes the improvement difference between tested method's score and a randomized mask score. We have five $\kappa \in [0, 1]$ KAR scores for a method. We have five $\rho \in [0, 1]$ ROAR scores for a method and five $z \in [0, 1]$ scores from a random mask condition. This corresponds to each saliency map spatial scale which we tested. We compute a simple difference score as:

$$D(\rho, \kappa) = \sum_{p=1}^5 (z_p - \rho_p) + \sum_{q=1}^5 (\kappa_q - z_q) \quad (8)$$

The *second method* is an information gain score given by:

$$I(\rho, \kappa) = - \sum_{p=1}^5 \rho_p \cdot \log_2 \left(\frac{\rho_p}{z_p} \right) - \sum_{q=1}^5 z_q \cdot \log_2 \left(\frac{z_q}{\kappa_q} \right) \quad (9)$$

Table 1 shows the results.

3.2 Comparison With Popular Methods

We compare our method with three popular saliency map methods using the standard ROAR/KAR methodology. These are *Gradient Heatmaps* [22], *Guided Backprop* [25] and *Integrated Gradients* [26]. We also include conditions with all three using *SmoothGrad-Squared* [24] which generally gives the best results. We create a GradCAM++ [2] comparison by creating a CAM saliency map based on the maximum class activation logit and not the ground truth class. Creating a saliency map based on the ground truth would give it a rather unfair advantage.

We compare three different weighting strategies when combining the saliency maps from all five scales. In the first strategy, we weight all five maps equal [1,1,1,1,1]. In the second, we use a rule-of-thumb like approach where we weight the first layer the least since it should be less accurate. Then each successive layer is weighted more. For this we choose the weights to be [1,2,3,4,5]. The third method weights the maps based on the expected accuracy given our results when we computed Table 1. These prior weights are [0.18, 0.15, 0.37, 0.4, 0.72]. The reason for showing the rule-of-thumb results is to give an idea of performance given imperfect weights since one may not want to spend time computing optimal prior weights. To fairly compare the three popular saliency map methods with our own, we adopt a methodology as close as possible to [8]. We train a

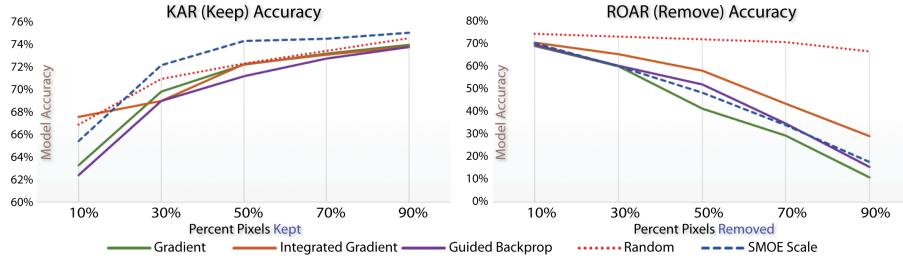


Fig. 5. SMOE Scale with prior layer weights is compared with three popular baseline methods that all use Squared SmoothGrad. Scores for these three are taken from [8]. The Y-axis is the **model accuracy** on *ImageNet* only. The X-axis is how much of the input image **salient pixels are kept or removed**. KAR keeps the most salient locations. Higher accuracy values are better. ROAR removes the most salient regions. Lower values are better. Our method does not seem to suffer as much when the 10% least salient parts are removed in KAR and in general maintains a better score. Our ROAR scores are very similar to Guided Backprop. Finer grain details can be seen in *Appendix* Table 5. Note that these results are *not* per layer. For a closer numerical comparison with the mask layer method, see *Appendix* Table 4.

ResNet-50 from scratch on ImageNet with either ROAR or KAR masking (computed by each of the different saliency mapping approaches in turn) at the start of the network. Otherwise, our training method is the same as [7]. The comparison results are shown in Figure 5. We can try and refine these results into fewer scores by subtracting the sum of the ROAR scores from the sum of the KAR scores. The results can be seen in Table 2. The KAR score for our method is superior to all three gradient methods. The ROAR score is better than Guided Backpropagation and Integrated Gradients. This suggests our method is superior at correctly determining which locations are *most* salient, but is not as good as Gradient Heatmaps for determining which parts of the image are *least* informative. We improve the results without sacrificing much efficiency by combining SMOE Scale with GradCAM++. This is done by simply multiplying the two maps together pixelwise. If we invert the CAM map by subtracting it from 1 and then multiply with SMOE Scale, this gives us a *Non-Class* map. These are things the network found interesting, but are not part of the class decision. An example of the combination can be seen in figure 6.

4 Discussion

4.1 Method Efficiency

The method as proposed is much faster than the three gradient baseline comparison methods. Given a ResNet-50 network, we only process five layers. The other methods require a special back propagation step over all layers. We can compute the cost in time by looking at operations that come from three sources. The first is the computation of statistics on tensors in five layers. The second

Table 2. Combined KAR and ROAR scores for several methods on ImageNet Only. The top three rows show several popular methods with *Squared SmoothGrad* [24]. These scores are created by simply summing the individual scores together. ROAR is negative since we want it to be as small as possible. *Prior Layer Weights* means we applied layer weights based on the prior determined accuracy of the layer saliency map. We include our top three scoring methods. The SMOE Scale method outperforms the three fine resolution gradient methods that use SmoothGrad on KAR. It outperforms Guided Backprop and Integrated Gradients on ROAR as well. Grad-CAM++ does surprisingly well on its own even though its saliency map has a rather weak definition. Truncated normal entropy scores a little worse than SMOE Scale if all scales have the same weight. Since SMOE Scale gains its largest performance boost in the earlier layers, when we apply prior weighting, we reduce that advantage. The best score is obtained by combining SMOE Scale and GradCAM++. A column shows the speed here as the time a method should be expected to take in orders of magnitude (e.g. *medium* should be a base 10 order of magnitude slower than *fast* and so on). More details can be seen in *Appendix* Table 5.

Method	KAR	ROAR	SUM	Speed	Resolution
Integrated Grad [26]	3.62	-3.58	0.03	Slow	Fine
Gradient [22]	3.57	-3.54	0.04	Medium	Fine
Guided Backprop [25]	3.60	-3.57	0.04	Medium	Fine
GradCAM++ [2]	3.64	-2.27	1.37	Fast	Coarse
Trunc Normal Ent + Layer Weights [1,1,1,1,1]	3.61	-2.47	1.14	Fast	Fine
SMOE Scale + Layer Weights [1,1,1,1,1]	3.62	-2.46	1.15	Fast	Fine
SMOE Scale + Layer Weights [1,2,3,4,5]	3.62	-2.34	1.28	Fast	Fine
Normal Std + Prior Layer Weights	3.61	-2.32	1.29	Fast	Fine
Trunc Normal Ent + Prior Layer Weights	3.61	-2.31	1.30	Fast	Fine
SMOE Scale + Prior Layer Weights	3.61	-2.31	1.30	Fast	Fine
Integrated Grad -w- SmoothGrad Sq.	3.56	-2.68	0.88	Slowest	Fine
Guided Backprop -w- SmoothGrad Sq.	3.49	-2.33	1.16	Slow	Fine
Gradient -w- SmoothGrad Sq.	3.52	-2.12	1.41	Slow	Fine
SMOE Scale + Prior Wts. -w- GradCAM++	3.66	-2.22	1.44	Fast	Fine

is the normalization of each 2D saliency map. Third we account for the cost of combining the saliency maps.

Ops for our solution ranges from 1.1×10^7 to 3.9×10^7 FLOPs (using terminology from [7]) for a ResNet-50. The network itself has 3.8×10^9 FLOPs in total. The range in our count comes from how we measure *Log* and *Error Function* operations that are computationally expensive compared to more standard ops and whose implementations vary. We estimate the worst case from available software instantiations. Most of the work comes from the initial computation of statistics over activation tensors. This ranges from 9.2×10^6 to 3.7×10^7 FLOPs. In total, this gives us an overhead of 0.3% to 1.0% relative to a ResNet-50 forward pass.

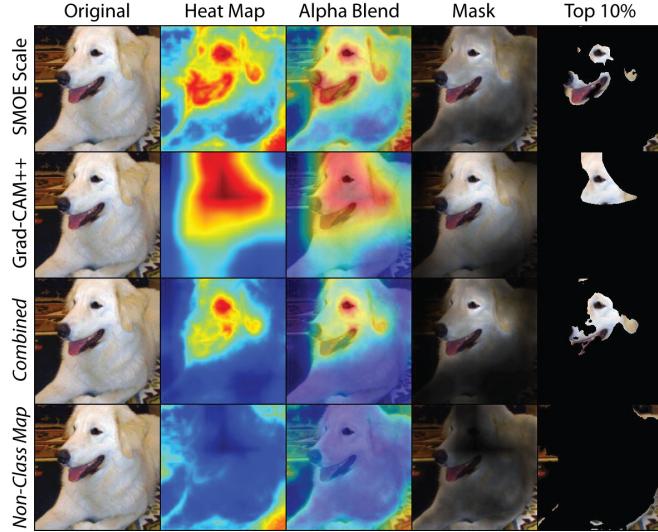


Fig. 6. Grad-CAM++ is combined with the SMOE Scale saliency map. This is done in the same way one would combine Grad-CAM with Guided Backprop. Here the output is from a ResNet-50 network trained on ImageNet. For larger networks, also including DenseNet, the Grad-CAM family solutions form an extremely coarse representation. Our method efficiently remedies this. The Non-Class map shows activations not related to the target class. This example is from ILSVRC2012 validation image 00049934. More examples can be seen in the *appendix* as figures 14 thru 28.

All gradient methods have a nominal overhead of at least 100%. A breakdown of the FLOPs per layer and component can be seen in Table 6 in the *appendix*.

Compared to any method which requires a full backward pass, such as gradient methods, our solution is nominally between $97x$ and $344x$ faster for non-SmoothGrad techniques, which according to [8] performs poorly on ROAR/KAR scores. We are between $1456x$ and $5181x$ faster than a 15-iteration SmoothGrad implementation that yields the competitive results in Table 2. 15 iterations as well as other parameters was chosen by [8] who describe this selection in more detail.

The memory footprint of our method is minuscule. Computation over tensors can be done inline which leaves the largest storage demand being the retention of 2D saliency maps. This is increased slightly by needing to store one extra 112×112 image during bilinear up-sampling. Peak memory overhead related to data is about 117 *kilobytes* per 224×224 input image.

4.2 SMOE Scale is The Most Robust of the Efficient Statistical Measures

SMOE Scale is the only metric without a failure case among all the statistics we tested. We define a failure case as one of:

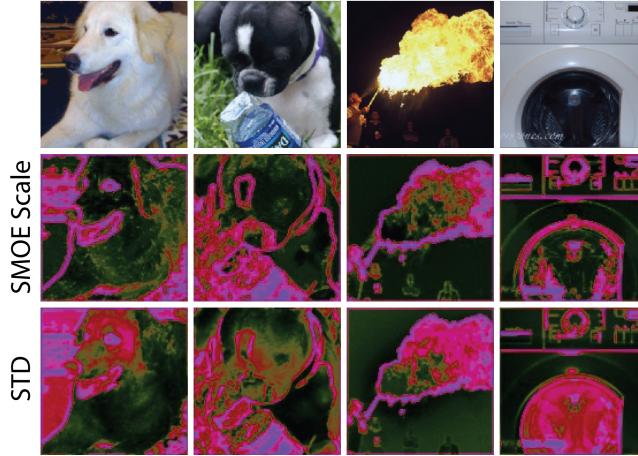


Fig. 7. These are examples of the first level saliency maps from SMOE Scale and Standard Deviation. It is common for both std and truncated normal entropy to problematically flood in areas with modest texture. This may explain why difference scores for these two methods are at or below the performance of a random saliency map.

- **1:** An information or difference score that ranks in the bottom quartile for a condition.
- **2:** An information or difference score at or below zero for a given condition. This means it can be worse than a random saliency map.
- **3:** A visually noticeable low score for a condition.

All statistics tested with the exception of SMOE Scale fail with case 1 and 3. Shannon entropy does not fail with case 2 and is the only one along with SMOE Scale which passes this one. However, it ranks fourth by difference score and fifth by information score.

SMOE Scale is the most *robust* statistic to use in terms of expected accuracy because it both scores the best and seems not to do poorly for any condition. The next highest scoring statistics, standard deviation and truncated normal entropy, are no better than a random map on layer 1. Figure 7 shows why this may be. It is important to note that layer 1 contains the finest pixel details and should be expected to be critical in rendering visual detail in the final combined saliency map. Full quartile rankings for the tested statistics can be seen in *Appendix* Table 7.

5 Conclusion

We have created a method of XAI saliency which is extremely efficient and is quantitatively comparable or better than several popular methods. When combined with a CAM method it is the most accurate method we have tested and

is still very fast. It can also be used to create a saliency map with interpretability of individual scale layers. Future work include expanding it to non-DNN architectures.

References

1. Chang, C.H., Creager, E., Goldenberg, A., Duvenaud, D.: Explaining image classifiers by adaptive dropout and generative in-filling. In: ICLR (2018)
2. Chattopadhyay, A., Sarkar, A., Howlader, P., Balasubramanian, V.N.: Grad-cam++: Improved visual explanations for deep convolutional networks. In: WACV (2018)
3. Choi, S.C., Wette, R.: Maximum likelihood estimation of the parameters of the gamma distribution and their bias. *Technometrics* **11**, 683690 (1969)
4. Dabkowski, P., Gal, Y.: Real time image saliency for black box classifiers. In: NIPS (2017)
5. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: CVPR (2009)
6. Fong, R., Vedaldi, A.: Interpretable explanation of black boxes by meaningful perturbation. In: ICCV (2017)
7. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: arXiv:1512.03385 (2015)
8. Hooker, S., Erhan, D., Kindermans, P.J., Kim, B.: Evaluating feature importance estimates. In: NIPS (2019)
9. Huang, G., Liu, Z., Weinberger, K.Q., van der Maaten, L.: Densely connected convolutional networks. In: CVPR (2017)
10. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: ICML (2015)
11. Itti, L., Koch, C.: Computational modelling of visual attention. *Nature Neuroscience* **2**, 194203 (2001)
12. Itti, L., Niebur, E., Koch, C.: A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on Pattern Analysis & Machine Intelligence* **20**, 1254–1259 (1998)
13. Jeong, J., Shin, J.: Training cnns with selective allocation of channels. In: ICML (2019)
14. Joblove, G.H., Greenberg, D.: Color spaces for computer graphics. *Computer Graphics* **12**, 20–25 (1978)
15. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: NIPS (2013)
16. Mundhenk, T.N., Konjevod, G., Sakla, W.A., Boakye, K.: A large contextual dataset for classification, detection and counting of cars with deep learning. In: ECCV (2016)
17. Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: ICML (2010)
18. Omeiza, D., Speakman, S., Cintas, C., Weldemariam, K.: Smooth grad-cam++: An enhanced inference level visualization technique for deep convolutional neural network models. In: arXiv:1908.01224 (2019)
19. Patro, B.N., Lunayach, M., Patel, S., Namboodiri, V.P.: U-cam: Visual explanation using uncertainty based class activation maps. In: ICCV (2019)
20. Samek, W., Binder, A., Montavon, G., Lapuschkin, S., Müller, K.R.: Evaluating the visualization of what a deep neural network has learned. *IEEE Transactions on Neural Networks and Learning Systems* **28**, 2660–2673 (2017)

21. Selvaraju, R.R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., Batra, D.: Grad-cam: Visual explanations from deep networks via gradient-based localization. In: ICCV (2017)
22. Simonyan, K., Vedaldi, A., Zisserman, A.: Deep inside convolutional networks: Visualising image classification models and saliency maps. In: arXiv:1312.6034 (2014)
23. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: ICLR (2015)
24. Smilkov, D., Thorat, N., Kim, B., Vigas, F., Wattenberg, M.: Smoothgrad: removing noise by adding noise. In: arXiv:1706.03825 (2017)
25. Springenberg, J.T., Dosovitskiy, A., Brox, T., Riedmiller, M.: Striving for simplicity: The all convolutional net. In: ICLR Workshop (2015)
26. Sundararajan, M., Taly, A., Yan, Q.: Axiomatic attribution for deep networks. In: ICML (2017)
27. Tishby, N., Pereira, F.C., Bialek, W.: The information bottleneck method. In: arXiv:physics/0004057 (2000)
28. Welinder, P., Branson, S., Mita, T., Wah, C., Schroff, F., Belongie, S., Perona, P.: Caltech-UCSD Birds 200. Tech. Rep. CNS-TR-2010-001, California Institute of Technology (2010)
29. Yang, L., Luo, P., Loy, C.C., Tang, X.: A large-scale car dataset for fine-grained categorization and verification. In: CVPR (2015)
30. Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks. In: ECCV (2014)
31. Zhou, B., Garcia, A.L., Xiao, J., Torralba, A., Oliva, A.: Learning deep features for scene recognition using places database. In: NIPS (2014)

A Appendix

A.1 Derivation of SMOE Scale

The maximum likelihood estimator of scale in the Gamma probability distribution is given as:

$$\hat{\theta} = \frac{1}{kn} \sum_{i=1}^n x_i \quad (10)$$

This requires the additional iterative estimation of the shape parameter k starting with an estimate s :

$$s = \ln \left(\frac{1}{n} \sum_{i=1}^n x_i \right) - \frac{1}{n} \sum_{i=1}^n \ln(x_i) \quad (11)$$

Then we get to within 1.5% of the correct answer via:

$$k \approx \frac{3 - s + \sqrt{(s - 3)^2 + 24s}}{12s} \quad (12)$$

Then we use the Newton-Ralphson update to finish:

$$k \leftarrow k - \frac{\ln(k) - \psi(k) - s}{\frac{1}{k} - \psi'(k)} \quad (13)$$

But we can see application of Eqs 12 and 13 is monotonic. This is also apparent from the example which we can see in Figure 8.

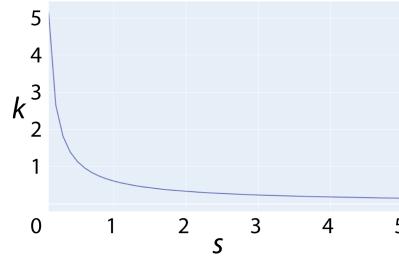


Fig. 8. A plot of the resulting k values from input s values in the gamma probability distribution maximum likelihood estimation. It is monotonic and reciprocal.

k is SMOE to $\frac{1}{s}$, so we rewrite Eq 10 with the reciprocal of k and optionally use the more efficient \log_2 as:

$$\hat{\theta}_{SMOE} = \frac{1}{n} \cdot \sum_{i=1}^n x_i \cdot \left[\log_2 \left(\frac{1}{n} \cdot \sum_{i=1}^n x_i \right) - \frac{1}{n} \cdot \sum_{i=1}^n \log_2(x_i) \right] \quad (14)$$

This then simplifies to:

$$\hat{\theta}_{SMOE} = \frac{1}{n} \cdot \sum_{i=1}^n \mu \cdot \log_2 \left(\frac{\mu}{x_n} \right) \quad (15)$$

We can see the results this gives with different kinds of data in Table 3

Table 3. Examples of SMOE Scale results given different data. This shows in particular when log variance and standard deviation give similar or diverging results. It is easier to see how SMOE Scale as a measure of variance is proportional to the mean. So, if we have lots of large values in an output, we also need them to exhibit more variance relative to the mean activation.

Input Values								Mean	STD	SMOE Scale
0.5	1	... 0.5	1	0.5	1	0.5	1	0.75	0.25	0.064
1	2	... 1	2	1	2	1	2	1.5	0.5	0.127
2	4	... 2	4	2	4	2	4	3	1	0.255
1	2	... 1	2	1	2	1	2	1.5	0.5	0.127
2	3	... 2	3	2	3	2	3	2.5	0.5	0.074
2	4	... 2	4	2	4	2	4	3	1	0.255
0.6125	1.8375	... 0.6125	1.8375	0.6125	1.8375	0.6125	1.8375	1.225	0.6125	0.254

A.2 More Examples of KAR Saliency Masks

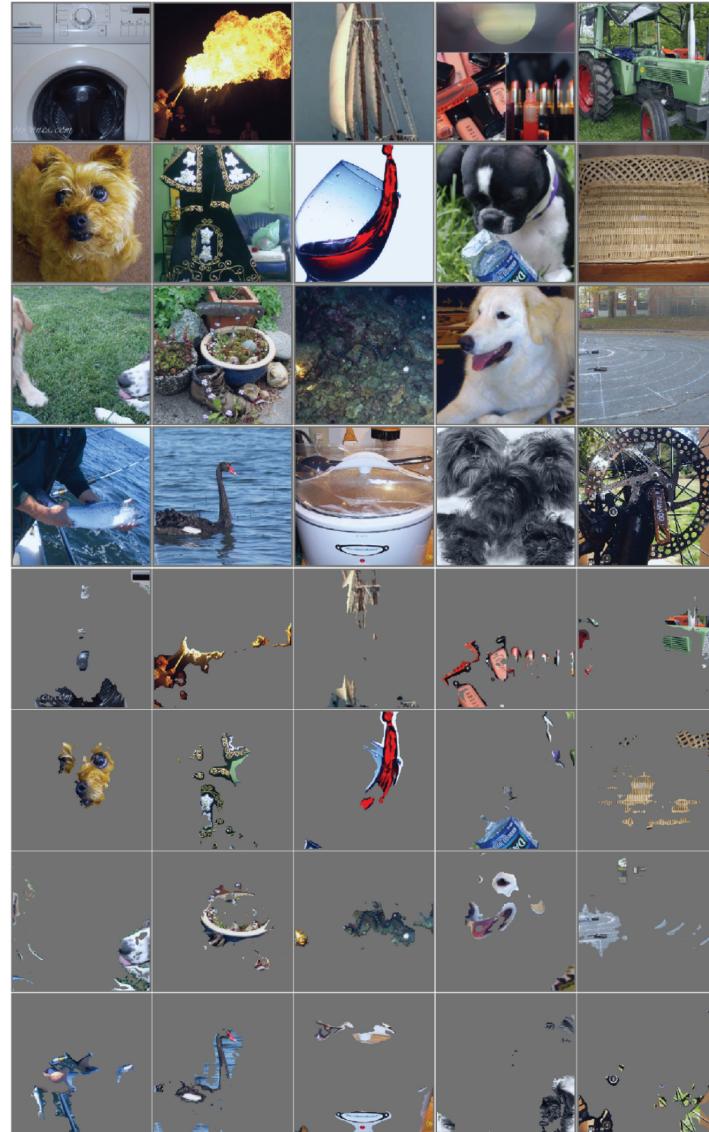


Fig. 9. These are the last mini-batch images in our *GPU:0* buffer when running the ImageNet validation set. The top images are the original input images and the ones on the bottom are 10% KAR images of the most salient pixels. These are images used when computing KAR scores.

A.3 More Examples of Combined Saliency Maps

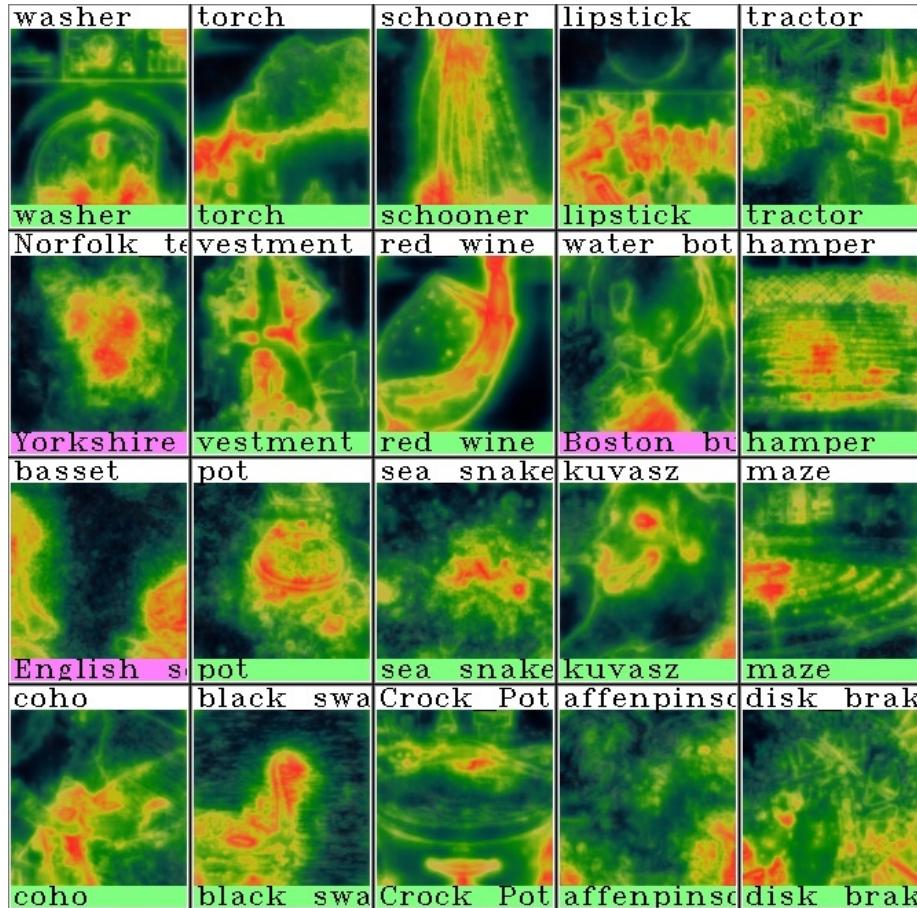


Fig. 10. These are more examples of combined saliency maps using the same images that appear in Figure 9. These images are *not* alpha blended with the original. Above each image is the ground truth label, while the label the network gave it is below. This was auto-generated by our training scripts.

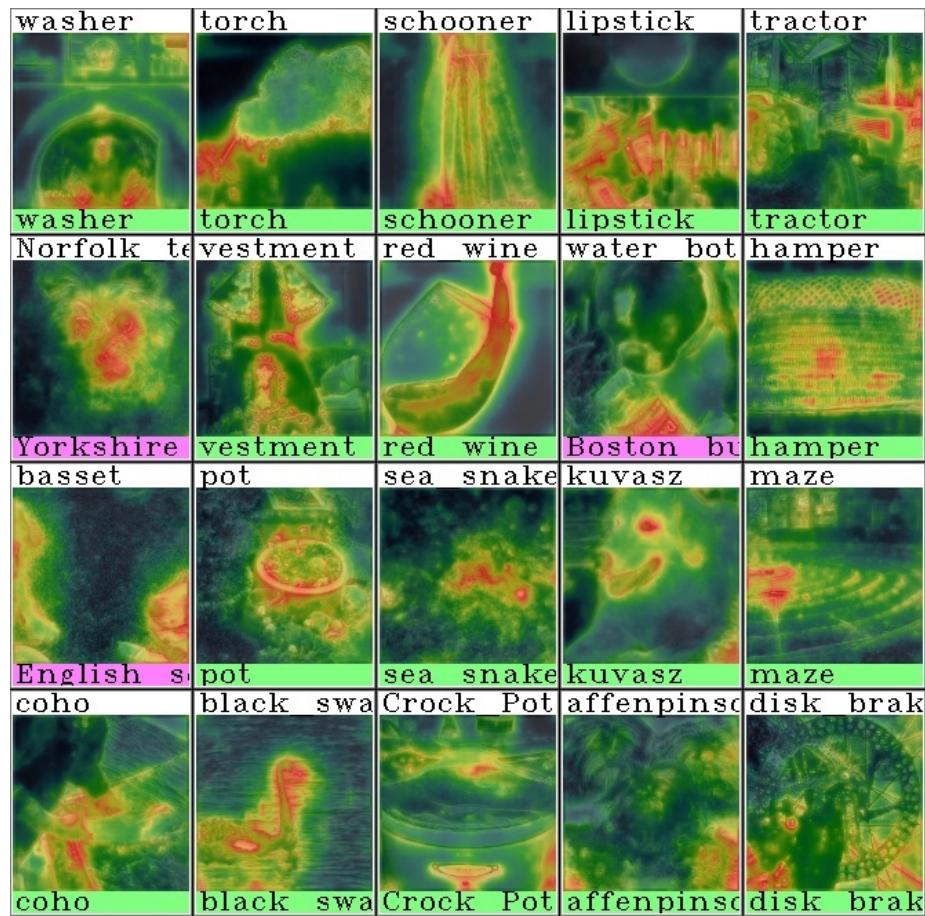


Fig. 11. These are the same as Figure 10 except with the original image gray scaled and alpha blended at 25%.

A.4 More Examples of LOVI Saliency Maps

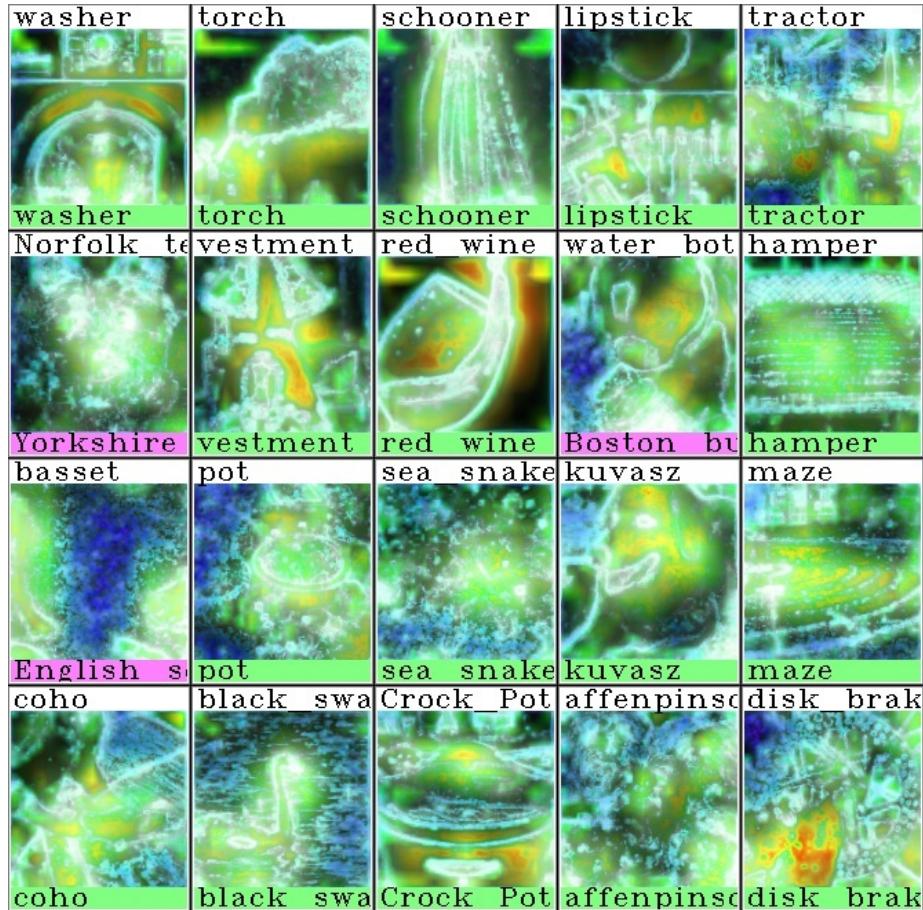


Fig. 12. These are more examples of visualizing multiple saliency maps using the same images that are in Figure 9. These images are *not* alpha blended with the original.

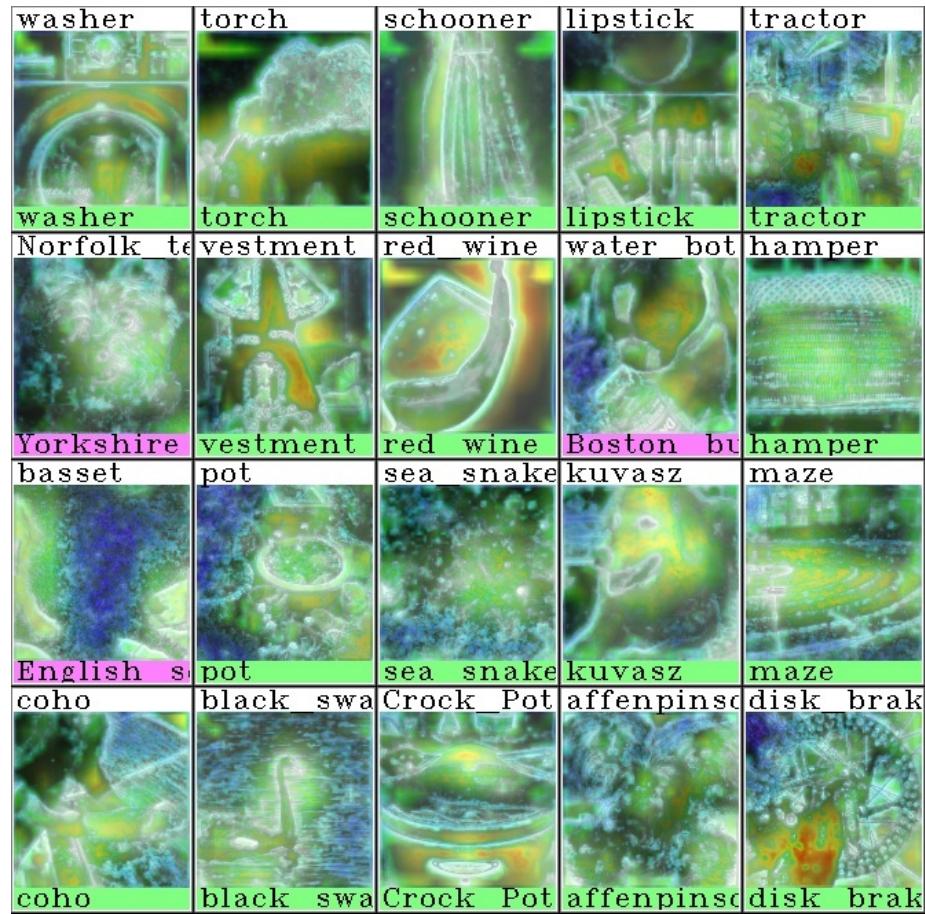


Fig. 13. These are the same as Figure 12 with the original image gray scaled and alpha blended at 25%.

A.5 Comparing Different Efficient Statistics in more Detail

This subsection shows the raw scores for each statistic over each dataset.

Table 4. KAR and ROAR results per dataset. This is a more detailed version of Table 1 and Figure 4. The differing effects of the distribution of data in the three sets seems to manifest itself in the L5 scores whereby the more concentrated the information is spatially, the better the ROAR/L5 score seems to be.

Method	KAR By Layer					ROAR By Layer				
	L1	L2	L3	L4	L5	L1	L2	L3	L4	L5
ImageNet										
Random	66.42	61.28	50.67	40.81	42.98	73.48	72.41	68.90	64.63	66.04
SMOE Scale	56.61	50.69	51.25	46.40	63.00	44.48	44.81	36.35	33.88	21.15
STD	51.84	50.73	50.72	46.16	62.82	45.78	42.74	36.17	34.41	22.88
Mean	53.21	40.34	50.88	46.85	62.56	52.66	64.10	37.85	34.15	19.19
Shannon Ent	55.43	45.69	50.89	47.17	61.18	44.74	51.38	38.73	35.78	18.07
Log Normal Mean	55.98	32.28	51.08	47.21	62.02	57.20	68.22	44.42	34.98	18.50
Log Normal ENT	53.01	42.52	51.13	46.85	62.26	47.92	62.64	38.73	34.50	18.91
Trunc Normal Mean	50.67	49.69	50.69	43.52	62.87	46.88	49.76	35.44	37.58	20.92
Trunc Normal Std	50.66	51.02	50.60	42.54	62.97	46.78	43.70	35.68	38.18	21.57
Trunc Normal Ent	50.84	50.62	50.57	43.63	62.97	46.92	45.48	35.56	37.64	21.25
Best	56.61	51.02	51.25	47.21	63.00	44.48	42.74	35.44	33.88	18.07
Worst	50.66	32.28	50.57	42.54	61.18	57.20	68.22	44.42	38.18	22.88
Places										
Random	57.20	53.59	47.83	41.26	39.45	60.77	60.25	58.14	56.41	55.26
SMOE Scale	49.76	45.67	46.39	40.57	53.50	44.35	44.61	39.80	40.26	27.94
STD	47.15	44.75	46.28	39.41	53.53	46.41	43.69	39.28	41.38	29.12
Mean	47.93	40.38	45.94	41.10	52.33	50.66	56.58	41.26	39.90	27.38
Shannon Ent	48.80	43.20	45.92	41.31	50.62	41.97	49.28	42.93	39.98	27.06
Log Normal Mean	50.05	35.87	46.23	41.45	51.67	52.21	57.91	45.17	39.73	26.88
Log Normal ENT	47.77	41.41	46.02	41.39	52.25	48.96	56.34	41.91	39.68	27.00
Trunc Normal Mean	46.25	44.76	46.12	38.08	53.18	46.92	46.92	48.83	42.09	28.06
Trunc Normal Std	45.96	45.35	46.38	37.61	53.38	46.41	46.76	44.86	42.43	28.68
Trunc Normal Ent	46.06	45.01	46.38	37.57	53.15	46.67	46.67	38.85	42.09	28.11
Best	50.05	45.67	46.39	41.45	53.53	41.97	43.69	38.85	39.68	26.88
Worst	45.96	35.87	45.92	37.57	50.62	52.21	57.91	48.83	42.43	29.12
COWC										
Random	65.05	57.43	52.30	64.31	65.55	77.38	75.44	71.11	78.25	77.32
SMOE Scale	64.19	63.02	71.05	62.87	80.65	45.16	44.09	43.97	62.78	59.49
STD	60.36	61.95	70.89	64.57	80.55	44.12	43.52	44.10	60.69	59.73
Mean	63.82	59.90	73.20	61.83	80.54	45.79	59.24	44.61	64.02	58.86
Shannon Ent	62.64	63.78	73.29	60.99	78.77	46.50	44.23	46.42	68.31	57.98
Log Normal Mean	66.37	46.05	72.89	60.21	80.38	48.98	71.02	46.69	67.19	58.16
Log Normal ENT	63.23	62.78	73.26	60.99	80.44	45.10	57.12	44.92	65.61	58.56
Trunc Normal Mean	60.00	63.35	72.08	65.62	80.54	42.98	44.44	44.15	61.60	59.45
Trunc Normal Std	59.52	63.74	71.58	65.58	80.54	42.76	43.45	43.98	62.07	59.81
Trunc Normal Ent	59.79	63.48	71.77	65.68	80.59	43.04	43.80	43.89	61.79	59.89
Best	66.37	63.78	73.29	65.68	80.65	42.76	43.45	43.89	60.69	57.98
Worst	59.52	46.05	70.89	60.21	78.77	48.98	71.02	46.69	68.31	59.89

A.6 Combined KAR and ROAR Scores with more Detail

This subsection shows the raw scores for each ROAR and KAR mask. We also added the non-SmoothGrad methods so we can see how much of an improvement it makes.

Table 5. Combined KAR and ROAR scores for several methods. This is a more detailed version of Table 2 and Figure 5. The top six rows show several popular methods with and without *Squared SmoothGrad* applied to give optimal results. These are taken from [8]. *Prior Layer Weights* means we applied layer weights based on the prior determined accuracy of the layer saliency map. We include our top three scoring methods. The SMOE Scale method outperforms the three baseline methods on KAR. It outperforms Guided Backprop and Integrated Gradients on ROAR as well as overall. The Gradient method is best overall, but as we discussed, it is much more expensive to compute.

Method	KAR Kept Percent					ROAR Removed Percent				
	10%	30%	50%	70%	90%	10%	30%	50%	70%	90%
Rand	63.53	67.06	69.13	71.02	72.65	72.65	71.02	69.13	67.06	63.53
Gradient	67.63	71.45	72.02	72.85	73.46	72.94	72.22	70.97	70.72	66.75
Guided Backprop	71.03	72.45	72.28	72.69	71.56	72.29	71.91	71.18	71.48	70.38
Integrated Grad.	70.38	72.51	72.66	72.88	73.32	73.17	72.72	72.03	71.68	68.20
Gradient -w- SmoothGrad Sq.	63.25	69.79	72.20	73.18	73.96	69.35	60.28	41.55	29.45	11.09
Guided Backprop -w- SmoothGrad Sq.	62.42	68.96	71.17	72.72	73.77	69.74	60.56	52.21	34.98	15.53
Integrated Grad. -w- SmoothGrad Sq.	67.55	68.96	72.24	73.09	73.80	70.76	65.71	58.34	43.71	29.41
SMOE Scale + Prior Layer Weights	65.44	72.14	74.28	74.51	75.01	70.40	60.33	48.48	34.23	17.72
SMOE Scale + Layer Weights [1,...,5]	65.76	72.60	73.97	74.53	74.94	70.28	60.93	48.73	35.66	18.01
SMOE Scale + Layer Weights [1,...,1]	66.13	72.28	73.72	74.52	74.97	71.28	63.58	52.85	38.74	19.72
Normal Std + Prior L. Weights	65.48	72.17	73.93	74.62	74.67	69.98	60.39	48.75	34.63	18.13
Trunc Normal Ent + Prior L. Weights	65.45	72.38	74.10	74.40	74.75	69.85	60.08	48.05	34.32	18.37
Trunc Normal Ent + Weights [1,...,1]	65.35	72.02	73.78	74.54	74.99	69.67	63.80	53.52	39.56	20.51
GradCAM++	66.75	73.53	74.52	74.86	74.70	70.88	60.79	48.06	32.53	14.92
SMOE Scale -w- GradCAM++	67.91	73.33	74.54	74.82	75.21	70.20	59.19	46.05	31.64	14.99

A.7 Operations Computation

Table 6. FLOPs for each layer. This is the breakdown of FLOPs for each layer. *Log* and *Error Function* are counted as one operation in this example. **SMOE Ops** is how many operations it takes to compute the initial saliency map using the SMOE Scale statistic. **Norm Ops** is the number of operations needed to normalize the saliency map. **Combine Ops** is the number of ops needed to upsample and combine each saliency map.

Layer	Layer Dimensions			FLOPS		
	Channels	Size H	Size W	SMOE Ops	Norm Ops	Combine Ops
Layer 1 64	112	112	3223808	150528	225792	
Layer 2 256	56	56	3214400	37632	338688	
Layer 3 512	28	28	1606416	9408	338688	
Layer 4 1024	14	14	803012	2352	338688	
Layer 5 2048	7	7	401457	588	338688	
Total			9249093	200508	1580544	

A.8 Cumulative Quartile Ranking of Different Efficient Statistics

Table 7. Cumulative quartile ranking over the three datasets. Here we have ranked the nine statistical methods over the 15 different conditions that are the same basis as for figure 4 and table 1. The *Difference Score* shows the results using Eq 8. The *Information Score* uses Eq 9. This shows what percentage of conditions a method ranked in the top n percent of scores. So for instance, we can see that SMOE Scale never scores in the bottom quartile for any condition using either scoring methods. Log Normal Entropy scores much better by this metric than it does by the prior metrics discussed. This may be because it never does especially well when there is a large difference in scores, but it tends to also never be the worst scoring method for any condition.

Method	Difference Score			Information Score		
	Top 25%	Top 50%	Top 75%	Top 25%	Top 50%	Top 75%
SMOE Scale	53.33%	93.33%	100.00%	60.00%	93.33%	100.00%
Log Normal Ent	40.00%	53.33%	80.00%	33.33%	53.33%	100.00%
Standard Dev	33.33%	46.67%	80.00%	40.00%	60.00%	93.33%
Trunc Normal Ent	33.33%	66.67%	80.00%	33.33%	66.67%	73.33%
Shannon Ent	33.33%	46.67%	73.33%	26.67%	40.00%	66.67%
Normal Mean	33.33%	53.33%	66.67%	26.67%	40.00%	66.67%
Log Normal Mean	26.67%	33.33%	53.33%	26.67%	33.33%	33.33%
Trunc Normal Mean	26.67%	53.33%	86.67%	26.67%	60.00%	86.67%
Trunc Normal Std	20.00%	53.33%	80.00%	26.67%	53.33%	80.00%

A.9 More Fast-CAM Examples

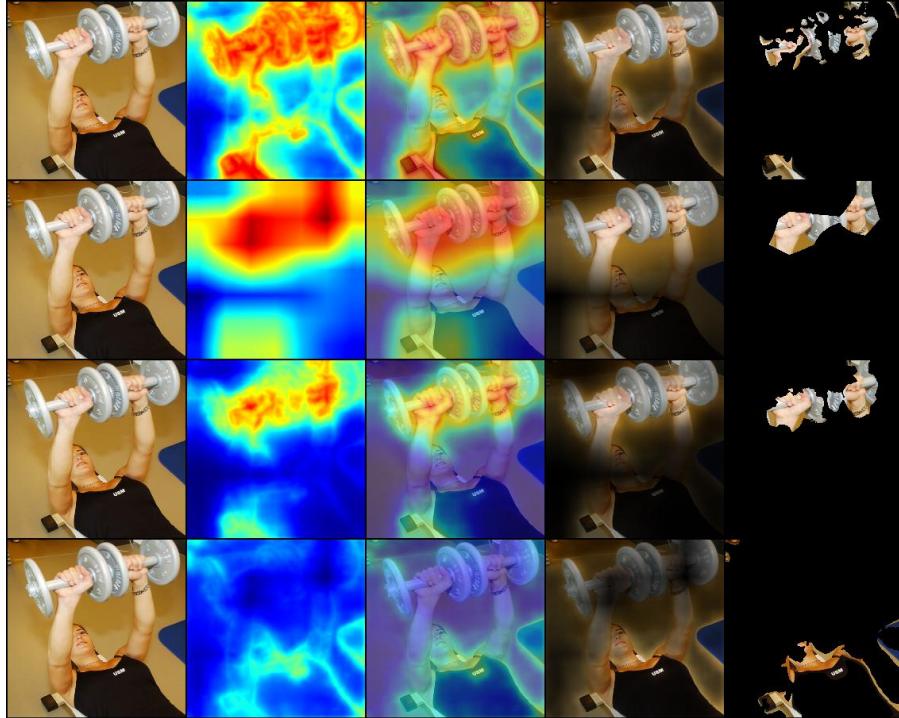


Fig. 14. This is another example of Fast-CAM in the same format as figure 6. This is from image ILSVRC2012 val 00049273. Ground truth label : *Dumbell* Network label : *Dumbbell*

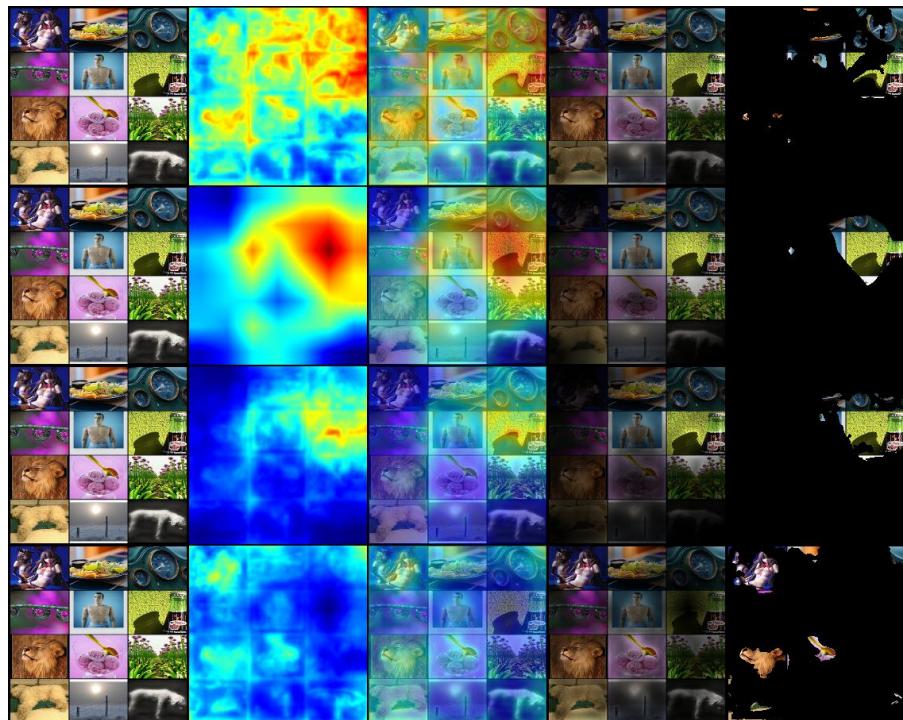


Fig. 15. This is another example of Fast-CAM in the same format as figure 6. This is from image ILSVRC2012 val 00049702. Ground truth label : *Ice Cream* Network label : *Web Page* **Comment** : The network appears to detect the ice cream as well as a few other classes. Web page seems like a more reasonable classification for this image.

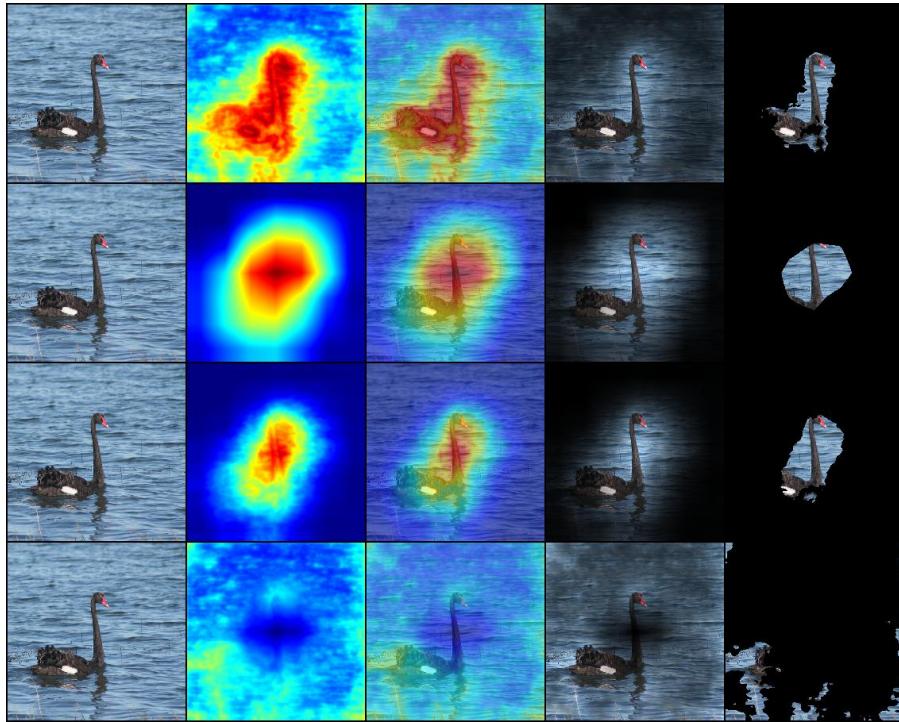


Fig. 16. This is another example of Fast-CAM in the same format as figure 6. This is from image ILSVRC2012 val 00049937. Ground truth label : *Black Swan* Network label : *Black Swan*

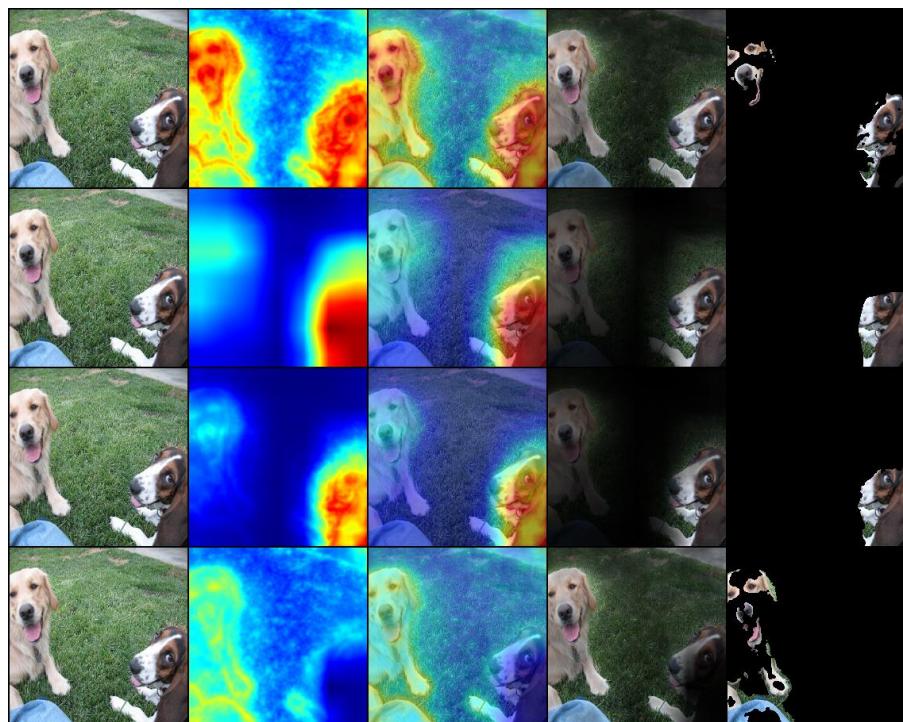


Fig. 17. This is another example of Fast-CAM in the same format as figure 6. This is from image ILSVRC2012 val 00049931. Ground truth label : *Basset* Network label : *Basset* **Comment** : given two different classes in this image, the network recognizes both but incidentally picks the correct label.

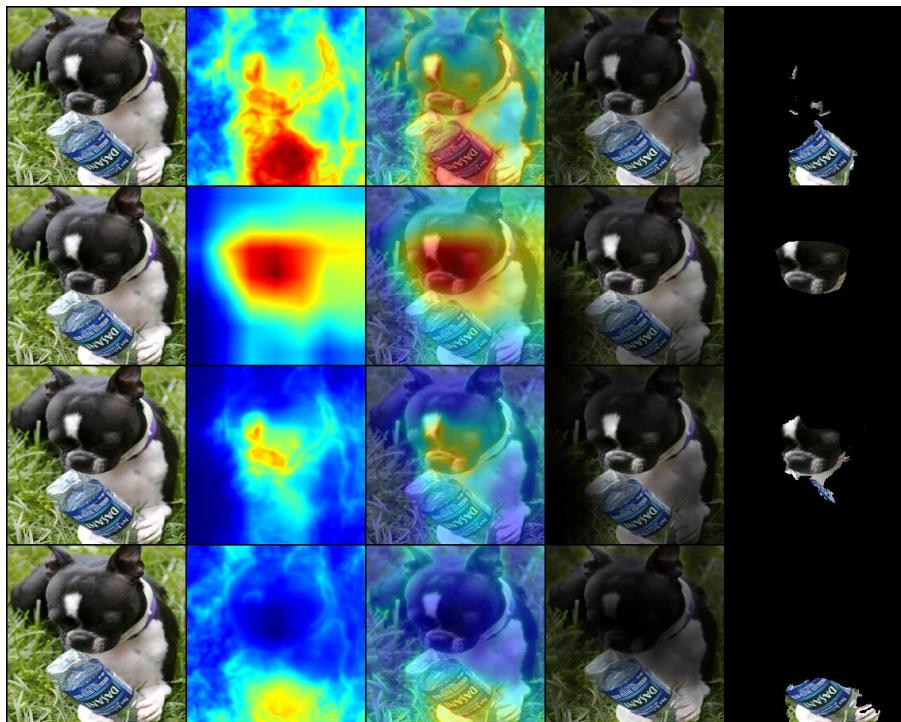


Fig. 18. This is another example of Fast-CAM in the same format as figure 6. This is from image ILSVRC2012 val 00049929. Ground truth label : *Water Bottle* Network label : *Miniature Schnauzer* **Comment** : There are two classes in this image. The network appears to recognize them both, but it picks the wrong class incidentally.

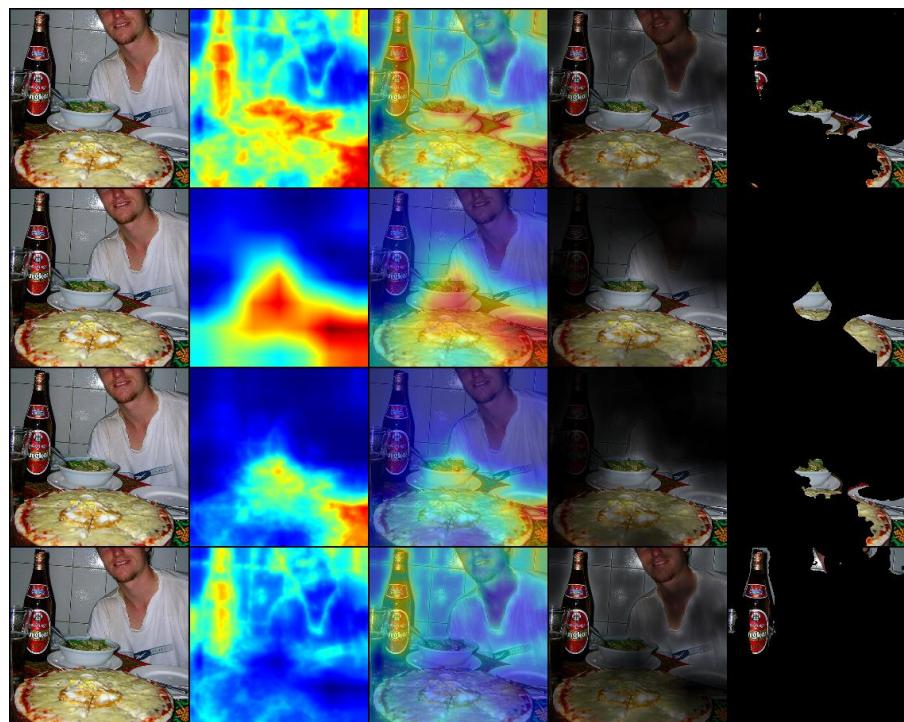


Fig. 19. This is another example of Fast-CAM in the same format as figure 6. This is from image ILSVRC2012 val 00049169. Ground truth label : *Pizza* Network label : **Menu Comment** : The pizza does not appear to be at all salient. The network does not notice it. Something makes this pizza unfamiliar to the network.

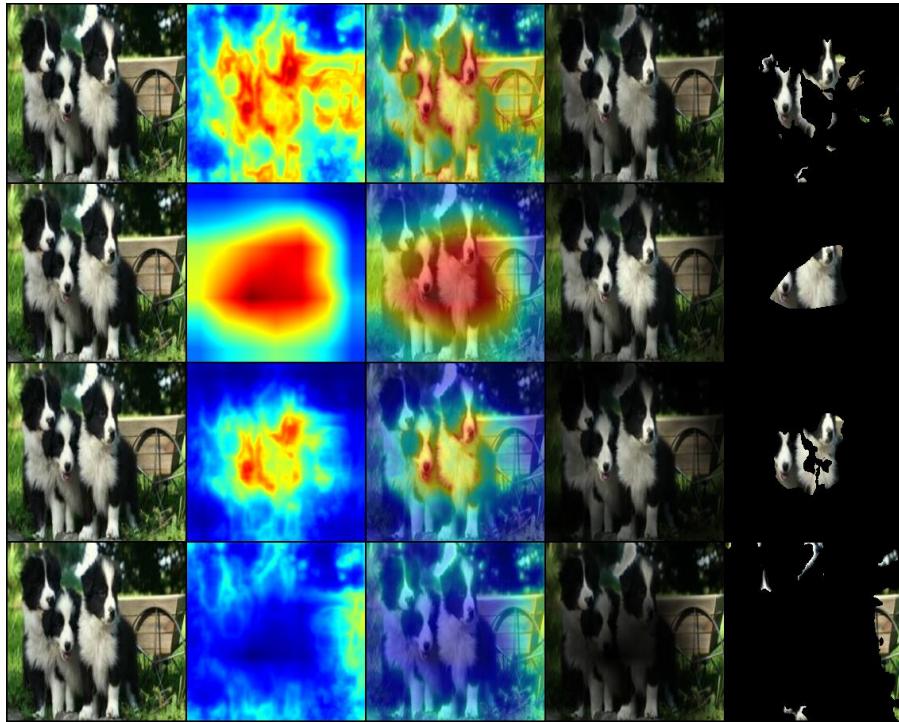


Fig. 20. This is another example of Fast-CAM in the same format as figure 6. This is from <https://pypi.org/project/pytorch-gradcam/> "collies.jpg". Network label : *Border Collie*

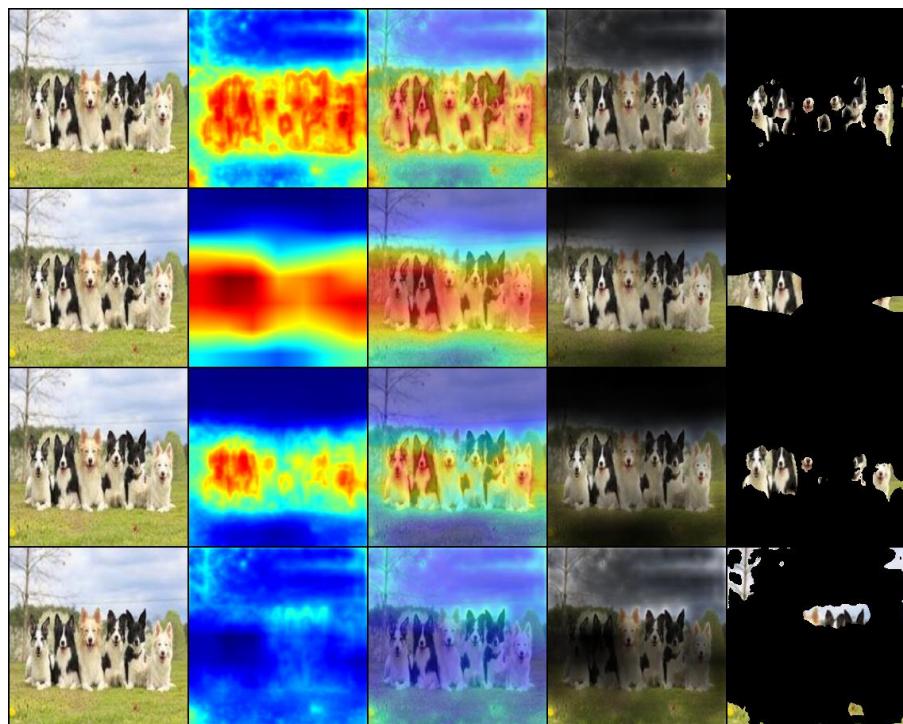


Fig. 21. This is another example of Fast-CAM in the same format as figure 6. This is from <https://pypi.org/project/pytorch-gradcam/> "multiple.dogs.jpg". Network label : *Border Collie* **Comment :** The middle three dogs are less salient than the ones on the outside.

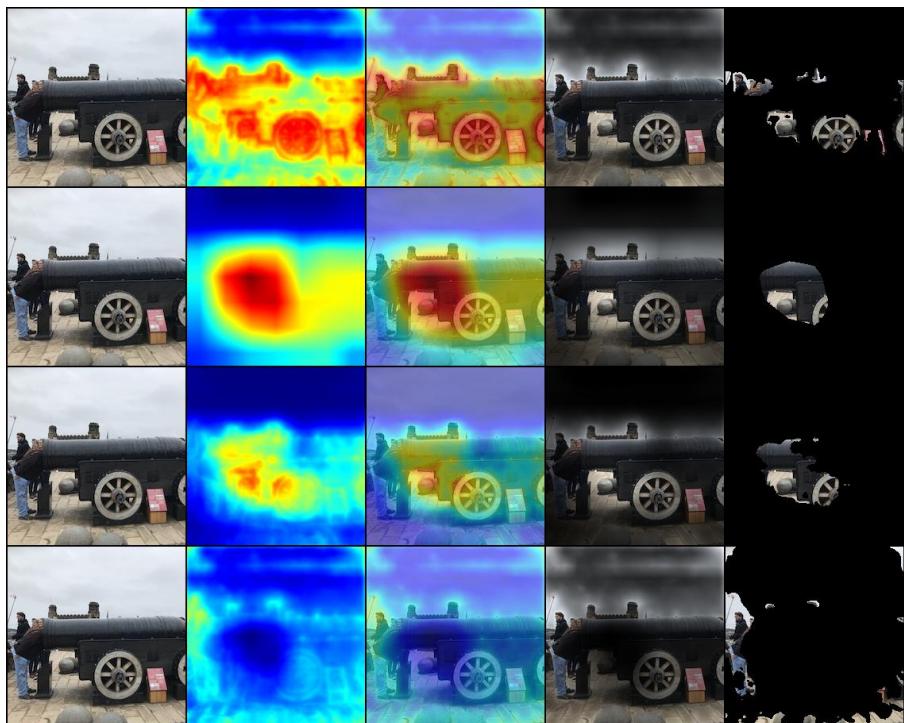


Fig. 22. This is another example of Fast-CAM in the same format as figure 6. This is from IMG_1382. Network label : *Cannon*

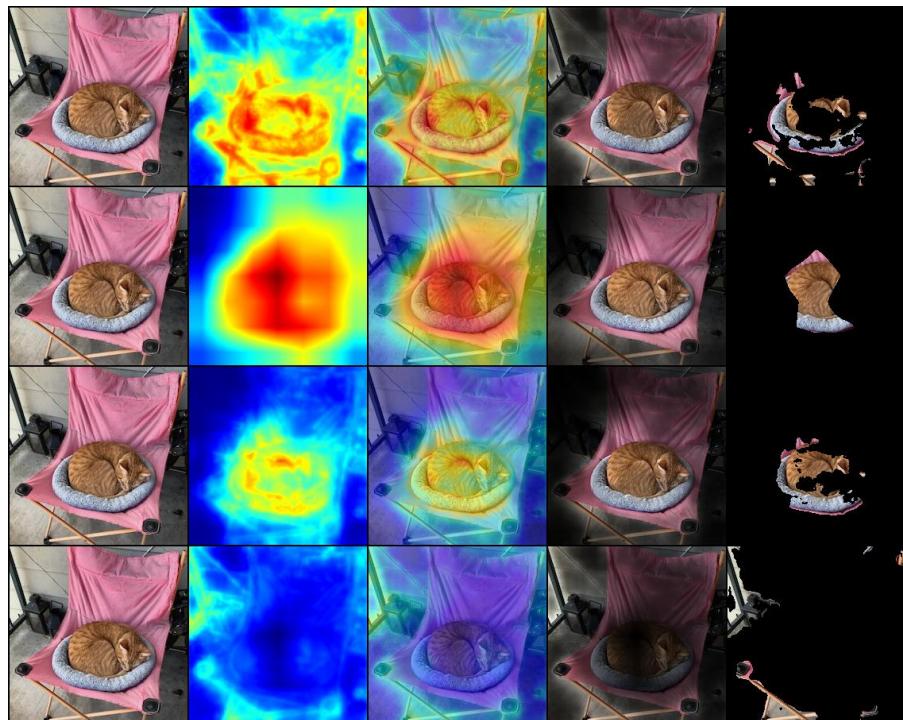


Fig. 23. This is another example of Fast-CAM in the same format as figure 6. This is from IMG_2730. Network label : *Sleeping Bag* **Comment** : A notable preponderance of sleeping bag images in ILSVRC have cats sleeping on them. Thus, the network may confound a cat as being a critical sleeping bag feature. We also would add that Miles is a happy cat.

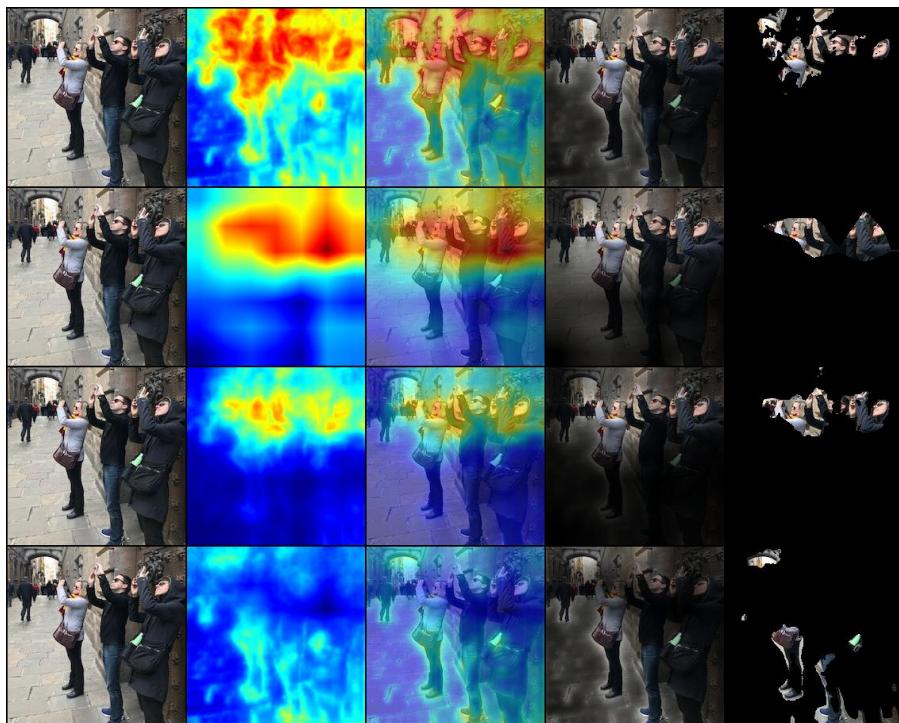


Fig. 24. This is another example of Fast-CAM in the same format as figure 6. This is from IMG_2470. Network label : *Abaya*

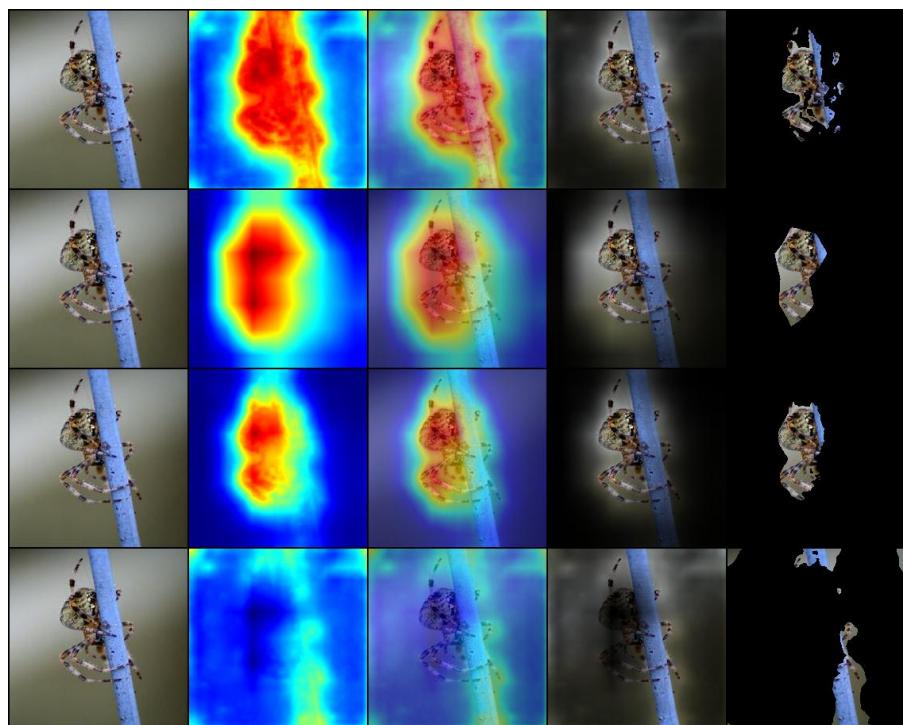


Fig. 25. This is another example of Fast-CAM in the same format as figure 6. This is a commonly used example image from the saliency literature.

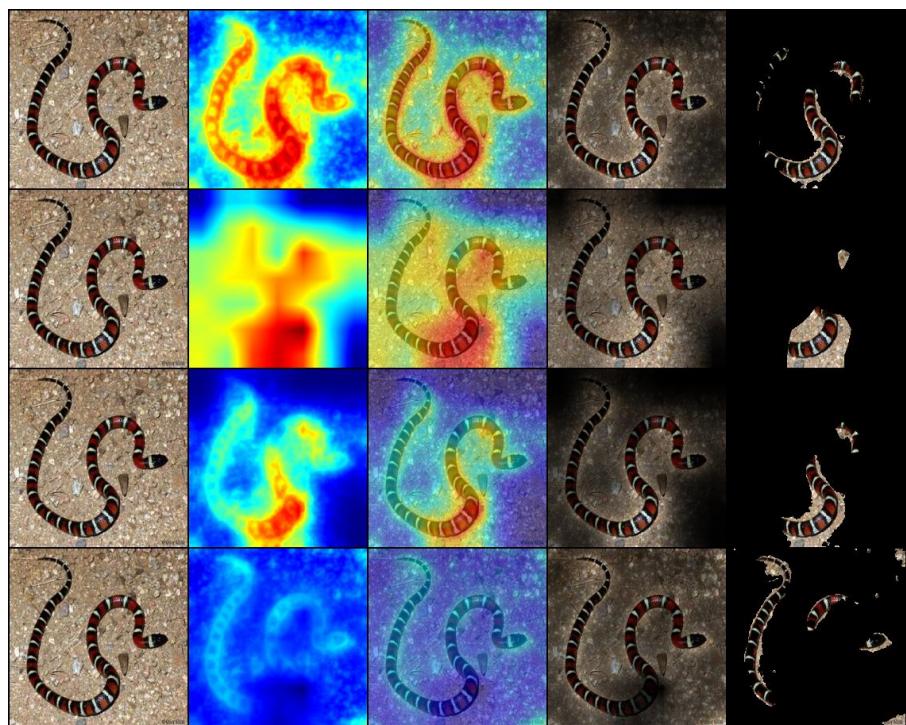


Fig. 26. This is another example of Fast-CAM in the same format as figure 6. This is a commonly used example image from the saliency literature.

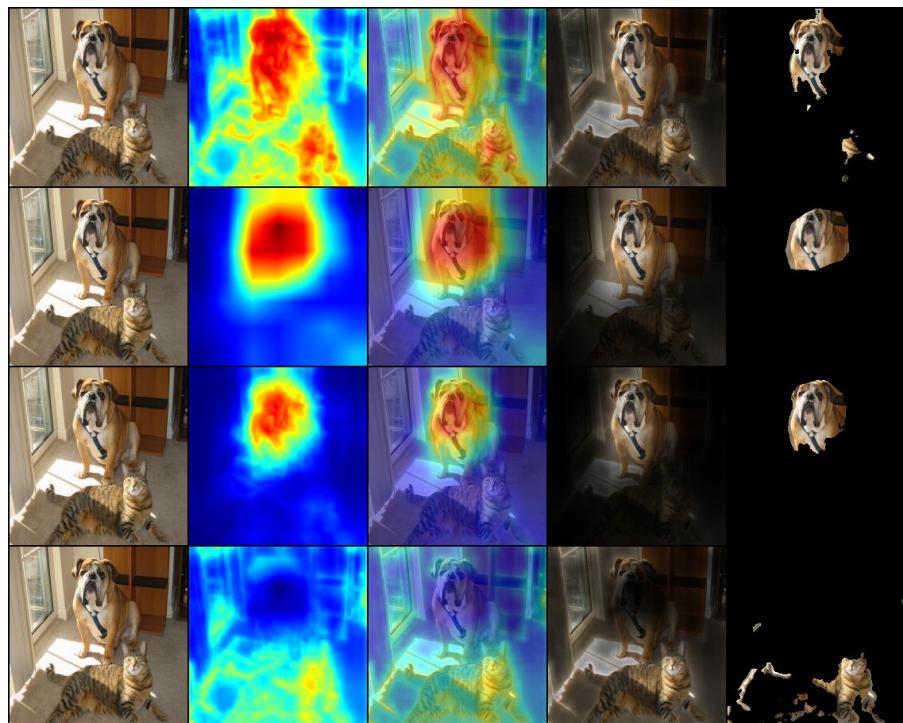


Fig. 27. This is another example of Fast-CAM in the same format as figure 6. This is a commonly used example image from the saliency literature.

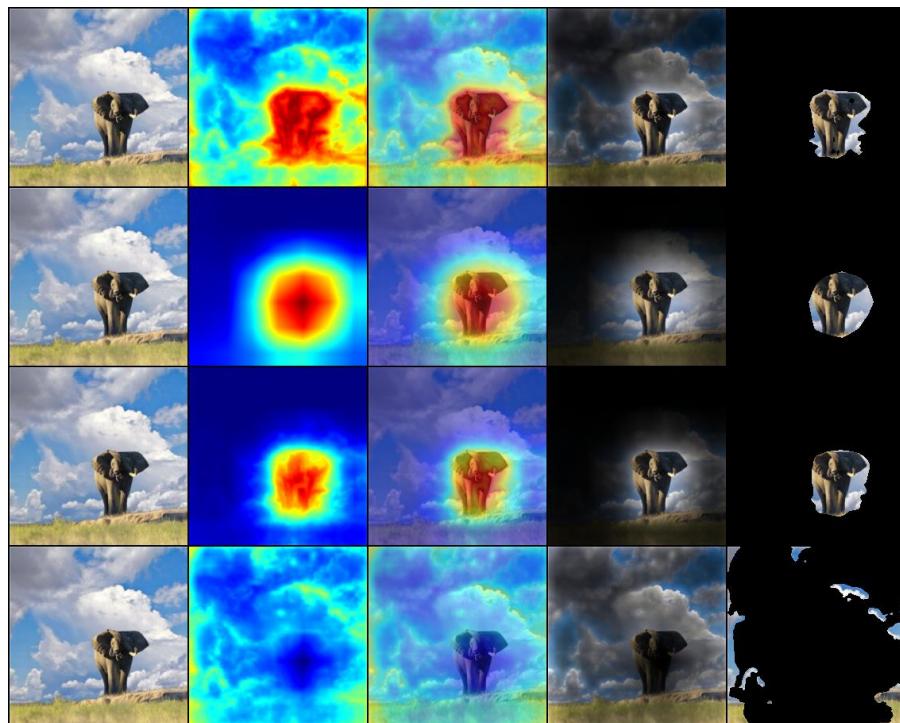


Fig. 28. This is another example of Fast-CAM in the same format as figure 6. This is a commonly used example image from the saliency literature.