ML4CV + XAI

Luca Domeniconi

February 18, 2025

1 Introduction

My work in this project is that of implementing and reproducing the experiments of Zheng et al. ([3])

2 Theoretical Background

Shap-CAM aims at developing a method to create saliency maps without relying on gradients and exploiting the game theory using shapley values.

2.1 Shapley Values

Shapley values is a mathematical tool to calculate the contribution of each player inside a game. Let's clarify this concept using an example.

2.1.1 Example[1]

Let's say there's a game with the following prizes for the first three players:

First	Second	Third
10.000	7.500	5.000

The game is played by two players Alice (A) and Bob(B). If they work together, they win the first prize. So we can write this as:

$$C_{A,B} = 10.000$$

where C_X is the value of the coalition X. How do Alice and Bob splits the prize? Shapley values can help us with that. The Shapley value of a player is the average marginal contribution of that player to all possible coalitions. In other words, it's how much a single player contribute to the final win.

Let's enumerate all the possible coalitions:

[1] Taken from this video

Coalition	Value
$C_{A,B}$	10.000
C_A	7.500
C_B	5.000
$C_\emptyset$	0

With these values we can calculate how much Alice increases the value of the coalition by joining it, also known as the marginal contribution.

$$C_{A,B} - C_B = 5.000$$
$$C_A - C_\emptyset = 7.500$$

Given these values, we can calculate the average marginal contribution of Alice to all possible coalitions as:

$$\phi_A = \frac{5.000 + 7.000}{2} = 6.250$$

This is exactly the saphley value of the player Alice. We can do the same calculation for Bob and obtain:

$$C_{A,B} - C_A = 2.500$$
$$C_B - C_\emptyset = 5.000$$
$$\phi_B = \frac{2.500 + 5.000}{2} = 3.750$$

Note: The sum of all the shapley values is always equal to the total value of the coalition with all the players, in this case $\phi_A + \phi_B = C_{A,B}$

2.1.2 Mathematical formulation

Consider a set of n players $\mathbb{P}$ and a function $f(\mathbb{S})$ which represents the worth of the subset of s players $\mathbb{S} \subseteq \mathbb{P}$. The function $f : 2^{\mathbb{P}} \to \mathbb{R}$ maps each subset to a real number, where $2^{\mathbb{P}}$ indicates the power set of $\mathbb{P}$. Shapley Value is one way to quantify the marginal contribution of each player to the result $f(\mathbb{P})$ of the game when all players participate. For a given player i, its Shapley value can be computed as:

$$\phi_i = \sum_{\mathbb{S} \subseteq \mathbb{P}, i \notin \mathbb{S}} \frac{(n - s - 1)!s!}{n!} [f(\mathbb{S} \cup \{i\}) - f(\mathbb{S})] \tag{1}$$

The Shapley value for player i defined above can be interpreted as the average marginal contribution of player i to all possible coalitions $\mathbb{S}$ that can be formed without it.

2.2 Computer Vision

When working with images, we can make an analogy with the example before and treat each pixel (input feature) as a player, and the final prize as the output of the whole network. Considering each pixel (i, j) of the activation of a convolutional layer $\mathbf{A}$ as the player in the cooperative game. Let $\mathbb{P} = \{(i, j)|i = 1, \ldots, h; j = 1, \ldots, w\}$ be the set of pixels in the feature map $\mathbf{A}$, where h, w stands for the height and the width of the feature map. Let $n = h \cdot w$ be the number of pixels in

the activation map. We then classify the worth function f as the class confidence Y^c, where c is the class of interest. For each subset $\mathbb{S} \subseteq \mathbb{P}, Y^c(\mathbb{S})$ denotes the class confidence of the input image when only the pixels in $\mathbb{S}$ remains and the other are "ignored". In general, models are not capable of "ignoring" input features, so to remove the contribution of a pixel (i, j), we set it to the average value of the whole feature map. The Shapley value of pixel (i, j) is then defined as:

$$\phi^c_{i,j,Shap-CAM} = \phi_{i,j}(Y^c) = \sum_{\mathbb{S}\subseteq\mathbb{P},(i,j)\notin\mathbb{S}} \frac{(n-s-1)!s!}{n!}[Y^c(\mathbb{S}\cup\{(i,j)\}) - Y^c(\mathbb{S})] \tag{2}$$

The obtained heatmap is then upsampled to the input image size to visualize the importance of each pixel, as it is done in Grad-CAM. Of course, if the the feature map we take into account is the input image itself, there's no need to upsample the heatmap as it already represents the importance of each pixel. To be noted that this approach is able to capture the interaction between different pixels as it consider all the possible subsets of pixels in the feature map. If we consider a simple example where two pixels improve the class confidence only if they are both present or absent and harm the confidence if only one is present, the equation 2 is able to capture this interaction.

2.3 Estimation of Shapley Values

Exactly computing Eq. (2) would require $O(2^n)$ evaluations. Intuitively, this is required to evaluate the contribution of each activation with respect to all possible subsets that can be enumerated with the other ones. To reduce the computational cost there are several methods. Zheng et al. uses a sampling algorithm which reduces the complexity to $O(mn)$, where m is the number of samples taken for each input pixel. For implementation details refer to the original paper [3].

3 SHAP Library[2]

The SHAP library is a Python library that implements several methods to compute Shapley values. The library is available at github. The library is well documented and provides a simple API to compute Shapley values for a given model. There are different algorithm to approximate Shapley values, the library provides a unified interface to use them. The library also provide a direct API for working with images and convolutional neural networks, which is the case of this project.

4 Problems

During the study of the work of Zheng et al. [3] I encountered some difficulties in understanding some details. The following are the things I didn't quite understand:

1. "For both the ImageNet and the PASCAL VOC datasets, all images are resized to 224×224×3". Given this statement and the fact that we calculate the Shapley values for each input pixel it follows naturally that the saliency map is naturally 224×224. The author then says "For a fair comparison, all saliency maps are upsampled with bilinear interpolate to 224×224.", suggesting that the calculated saliency map are smaller than 224×224. Also, given the Figure 1 from the same paper, the smoothness of the saliency maps of Shap-CAM suggest some type of upsampling. Maybe they are estimanting the Shapley values of some feature maps after convolutions instead of directly to the input image (???).

[2] After some experiments, captum and grad cam[2] library has been used instead, mainly for computational reasons

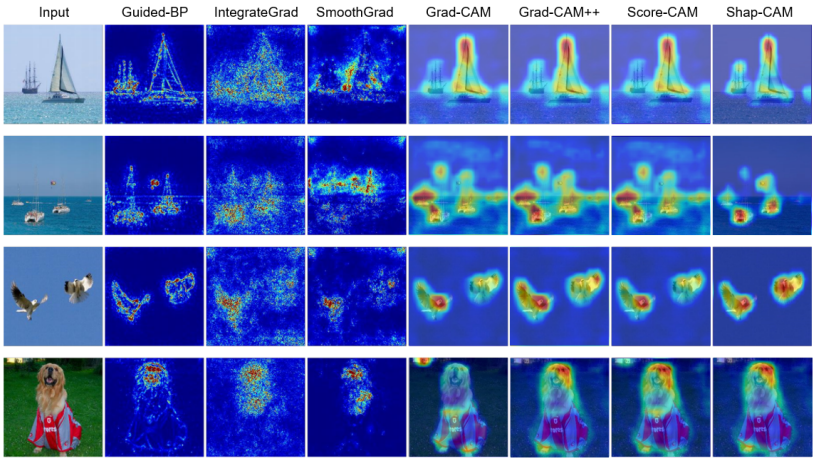


Figure 1: [**Taken from the paper**] Visualization results of Guided Backpropagation, SmoothGrad, IntegrateGrad, Grad-CAM, Grad-CAM++, Score-CAM and our proposed Shap-CAM.

2. "The original input is masked by point-wise multiplication with the saliency maps to observe the score change on the target class. In this experiment, rather than do point-wise multiplication with the original generated saliency map, *we slightly modify by limiting the number of positive pixels in the saliency map*". Not very clear how they actually use the produced saliency map.

3. In the definition of the introduced metrics, Average Drop and Increase in Confidence, the author defines the formula:

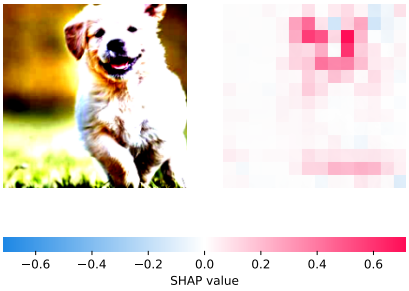
$$\text{Avg. Drop} = \sum_{i=1}^{N} \frac{\max(0, Y_i^c - O_i^c)}{Y_i^c} \times 100\%$$

   where Y_i^c refers to the prediction score on class c using the input image i and O_i^c refers to the prediction score on class c using the saliency map produced over the input image i. O_i^c is defined as the term "*saliency map*", which could refer to the point-wise multiplication with the original image or by *"slightly modify by limiting the number of positive pixels in the saliency map"* cited before.

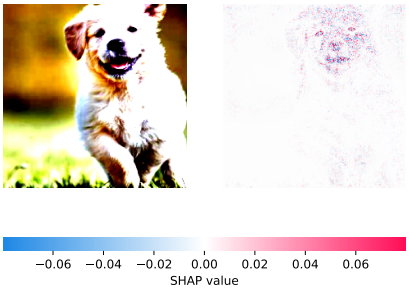
4. Regarding the model used, the author says "we use pre-trained VGG16 network from the Pytorch model zoo as a base model", probabily refering to this page. The problem is that the model is pretrained on ImageNet-1K, so the model outputs the probability distribution over 1000 classes. How did they use this model on classification task with the Pascal VOC 2007 datasets, that only contains 20 classes? They transformed the label from Pascal VOC ones to ImageNet-1K ones? They replaced the head of the classifier with a new one and then re-trained the model on Pascal VOC? If so, how was the re-training been done?

5 Doubts

I have the following doubts about calculating shapely values on an activation map instead of the input image. In Figure 2a I calculated the shap values of the layer `model.features[18]` and the



(a) SHAP values calculated on the layer `model.features[18]`.



(b) SHAP values calculated on the input image.

Figure 2: Example of SHAP values calculated using the DeepExplainer from the `shap` library.

results seems to make sense. My doubt is about why it should work. The plot on the right showing the shap values is red when a feature of the activation of the layer `model.features[18]` is important to the prediction of the correct class. But how can we be sure that the feature value at that position represents the input features near it? Maybe a feature in the upper right corner of that activation layer may have a big impact on the prediction, but how does this tells us that that feature actually represents the pixels in the upper right corner?
I think this can happen because of how convolutions works, but I think it's not always guaranteed. Maybe that's the reason the results in Figure 2a seems to make sense but it's not very good. It seems to me as we are mixing semantic information with spatial informations.
Nevertheless, looking at Figure 2b where the shap values are calculated relatively to the input image, the results are not very promising (I expected a smoother result).

6 Implementation

6.1 Model

For all the experiments, I used the VGG11 network present on PyTorch, pretrained on ImageNet. In order to use it with the PascalVOC dataset, I removed the original classification head and replaced it with one with only 20 classes in output and then did fine-tuning of the model on PascalVOC by keeping the model feature extractor frozen.

6.2 Attribution Methods

For the implementation I decided to compare the following 2 attribution methods:

- **DeepLiftShap** (from the captum library): *"Extends DeepLift algorithm and approximates SHAP values using Deeplift"*. From my understanding of this method, this should be mathematically equivalent to Shap-CAM.
- **GradCAM++** (from the pytorch-grad-cam libray[2]): differently from all the previous methods, this builds the attribution map starting from the gradient of the activations.

I avoided the use of the before cited SHAP library because the captum implementation is far more efficient and fast.

6.3 Metrics

For the two methods, I calculated the attribution map for each feature layers of the model and used it to calculate these 4 metrics:

- **Average Drop**[3]: When dot multipying the input image with the upsampled saliency maps, it measures how much the prediction on the correct class decreases. If it's 0, then it means that there's no chenge at all, so the saliency map has "selected" all the important pixels for the model. Lower is better.
- **Average Increase**[3]: When dot multipying the input image with the upsampled saliency maps, it measures how many times the prediction on the correct class increases. This can happen for example if the context of the image is distracting. Higher is better.
- **Deletion Curve AUC**[1]: remove the pixels 1% for each steps and measure the prediction on the correct class. Removing the pixels simply means setting its value to 0. The pixels are removed starting from the ones with a higher value in the saliency map (more important). In this way, if the saliency map correctly finds the important pixels, the prediction on the correct class should drop as soon as possible. It's possible to summarize this curve with its area underneath (AUC). Note that for this metrics *only the order* of the values in the saliency map are considered, and *not the actual values*. Lower is better.
- **Insertion Curve AUC**[1]: adds pixels 1% for each steps and measure the prediction on the correct class. Instead of starting from a total black image, the metric starts with a highly blurred version of the image. Adding a pixel simply means to set its value to the original one. The pixels are added starting from the ones with a higher value in the saliency map (more important). In this way, if the saliency map correctly finds the important pixels, the prediction on the correct class should rise as soon as possible. It's possible to summarize this

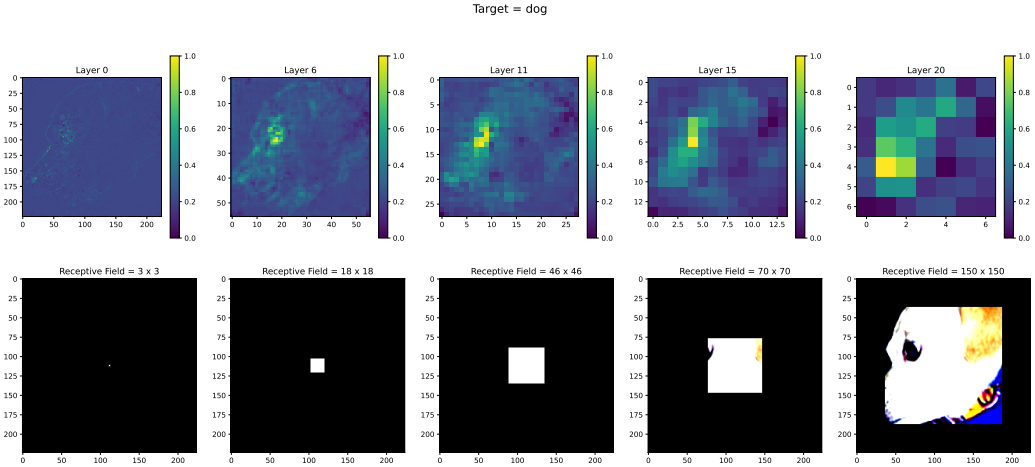


Figure 3: Visualization of the receptive field for some example activation maps.

curve with its area underneath (AUC). Note that for this metrics *only the order* of the values in the saliency map are considered, and *not the actual values*. Higher is better.

By saying "for each layer", I mean calculating the attribution map for the input of that particular layer. The results in Fig. 5 shows some particular pattern with respect to the layer type (Conv2D, ReLU or MaxPool2D) that I'm not really able to explain or justify.

The Shap-CAM paper[3] only provides average values for the Avr. Drop and Avr. Increase (red dotted line in Fig. 5), without specifying at which layer the saliency map is actually computed. For the Deletion Curve and Insertion Curve, only 2 qualitative results are shown, leaving no room for quantitative comparison.

6.4 Receptive Field

After each layer of the model, the receptive field (the part of the input image that influence a single element in an activation) gets bigger and bigger. Consequently, by going deeper into the network, each pixel in an activation map is representative of a bigger and bigger region of the input image.

By looking at Fig. 3 we can observe that, when using layer 20 for the creation of the saliency map, each pixels in the 7×7 matrix is directly influenced by a square of size 150×150 of the input image (almost 45% of the input image). In Fig. 4 we can see the size of the receptive field with respect to the layer depth.

7 Results

The results of these experiments are shown in Figure 5. By looking at the plot, it is very clear that the type of layer that we pick is related with the quality of the saliency map produces. In all the metrics, except for the "Deletion Curve AUC", using deeper layer to compute the saliency map gives better results.

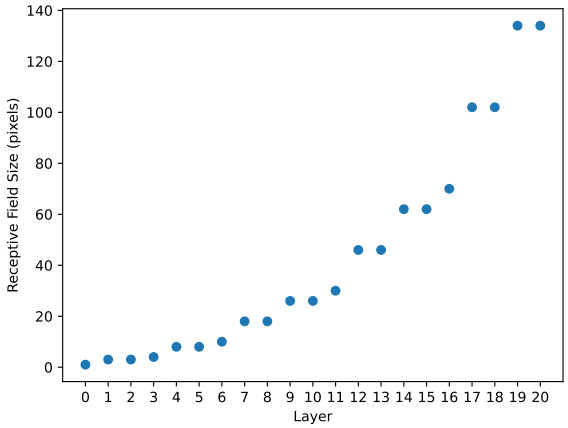


Figure 4: Size of the receptive field over the layer depth

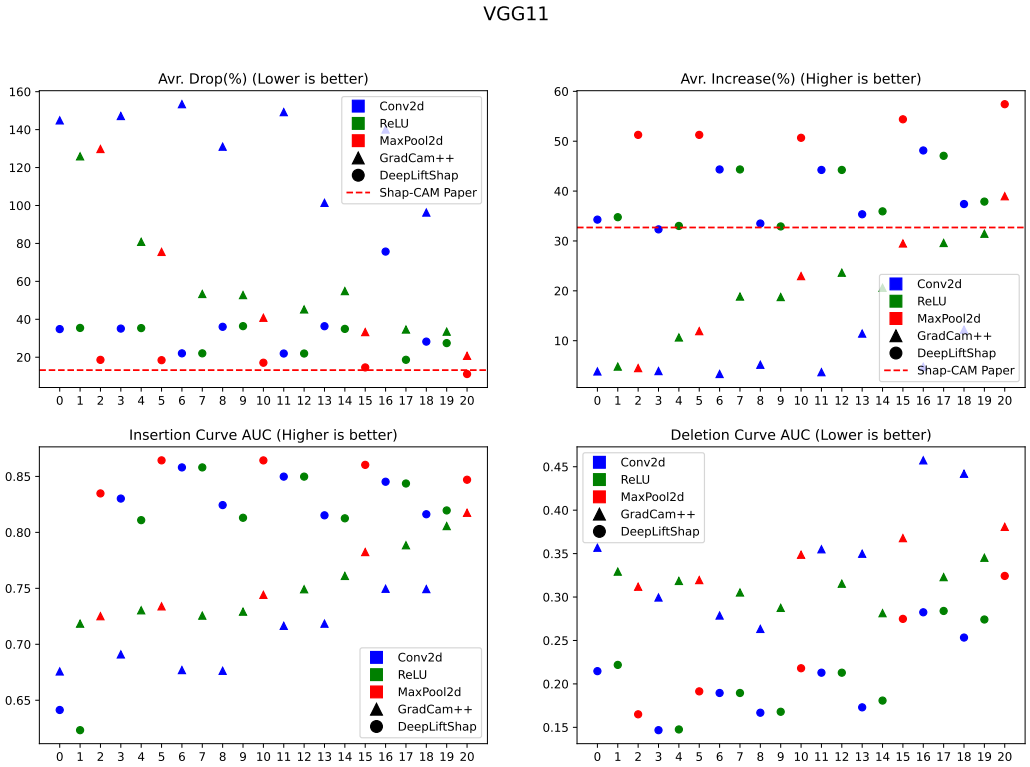


Figure 5: Metrics for each layer for the 2 attribution methods compared. Color represents layer type. Shape represent attribution method. For an explanation of these 4 metrics, see Section 6.3.

7.0.1 Paper comparison

For a comparison with the paper results, in the metrics "Average Drop" and "Average Increase", the DeepLiftShap methods performs better than Shap-CAM, especially in the "Average Increase" metric. Of course, for a fair comparison, we should compare the results of the metric by computing the saliency maps at the same layer.

8 Comparison Idea

When calculating the metrics for the saliency maps, certain metrics can be improved by simply biasing the attribution method into outputting bigger values for every pixel. For example, if we take a saliency map composed of all 1s, the "Average Drop" will be the best value possible for it, that is 0. Of course this is not what we want. The following is an idea to try to compare in the most fair way two different saliency maps.

The main idea, is to make the 2 saliencies to have the same sum of all values, but to keep untouched the maxima in the function. Given a saliency map $S(i,j) : \mathbb{R}^2 \to [0,1]$, the rescaled saliency map is defined as $S'(i,j) = S(i,j)^{\exp(\alpha)}$, where α can be found by solving the following optimization problem:

$$L(S(i,j),\alpha) = \left(\sum_{i,j} S(i,j)^{\exp(\alpha)} - (r \times H \times W) \right)^2$$

$$\alpha = \arg\min_{\alpha \in \mathbb{R}} L(S(i,j),\alpha)$$

where:

- H, W are the height and width of the saliency map.
- $r \in [0,1]$ is the rescale factor. In other words, define how much of the maximum total sum to keep.
- $\exp(\alpha)$ is just a way to keep the exponent greater than 0.

The optimal value for α can be easily found because the loss is convex. By normalizing 2 saliency maps with the same rescale factor r, the comparison is more fair. The results can be seen in Figure 6. To be noted is that they're very similar to the ones in Figure 5, just with a different scale on the y axis.

To measure quantitively the impact of the rescaling, Figure 7 shows that when using rescaling with $r = 0.3$, the difference in the results is closer to 0, hinting to the fact that these metrics are biased by the total sum of the saliency maps.

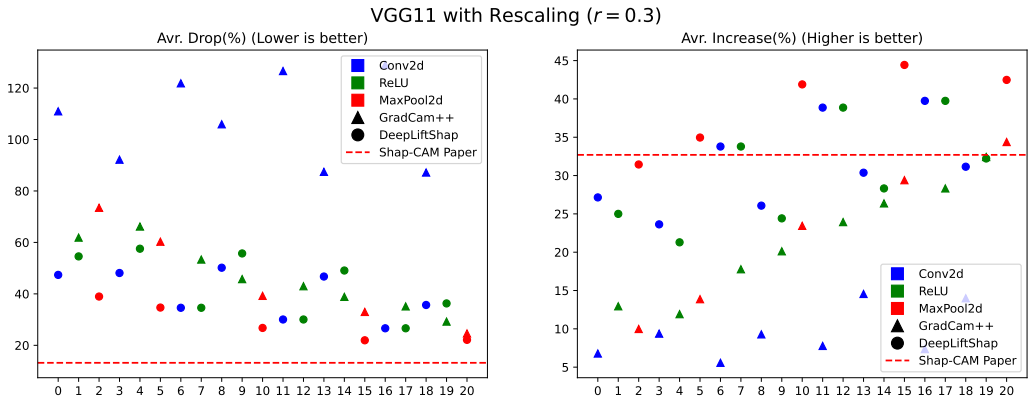


Figure 6: Results with rescaling done with $r = 0.3$. This means that the 2 saliency maps produced by the 2 methods have the same sum. Insertion Curve AUC and Deletion Curve AUC are not reported, as they remain the same (applying a monotonic function doesn't change the order of the values).

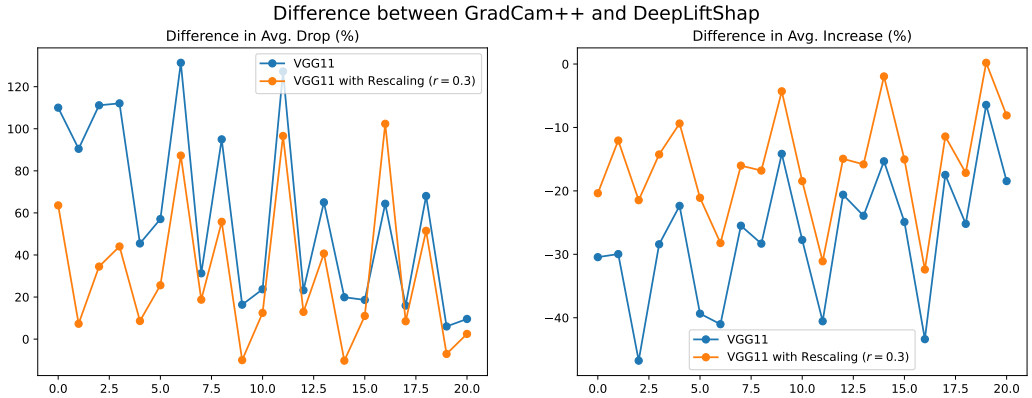


Figure 7: Difference between GradCam++ and DeepLiftShap results. When using scaling with $r = 0.3$, the difference in both metrics is closer to 0.

References

[1] Vitali Petsiuk, Abir Das, and Kate Saenko. "RISE: Randomized Input Sampling for Explanation of Black-box Models". In: *ArXiv* abs/1806.07421 (2018). URL: https://api.semanticscholar.org/CorpusID:49324724.

[2] Jacob Gildenblat and contributors. *PyTorch library for CAM methods*. https://github.com/jacobgil/pytorch-grad-cam. 2021.

[3] Quan Zheng et al. "Shap-CAM: Visual Explanations for Convolutional Neural Networks based on Shapley Value". In: *European Conference on Computer Vision*. 2022. URL: https://api.semanticscholar.org/CorpusID:251402673.