Ranking Explainable AI Methods Using Pixel-Level Evidence in Classification Tasks

Anh Khoa Pham
Group vibe_coding_scientist
https://github.com/khoaphamanh/project_iml

1 Motivation

In the Interpretable Machine Learning course, we have become familiar with gradient-based methods such as Saliency Maps and Integrated Gradients, which can be applied to virtually any deep learning model.

In section 2.3, we introduce Gradient-weighted Class Activation Mapping, a local explanation method specifically designed for Convolutional Neural Networks (CNNs) to explain a particular model decision for a given image input.

At the same time, within the image classification setting, to enhance transparency, we consider not only categorical image labels but also pixel-level ground-truth annotations, which are used to evaluate the faithfulness of the three explanation methods mentioned above, namely Gradient-weighted Class Activation Mapping (GC), Saliency Maps (SM), and Integrated Gradients (IG).

2 Related Topics

SM and IG have long been two well-known local explanation methods, and they can be applied to any deep learning neural network through gradients [1]. GC can also be implemented using deep neural networks and gradients. However, it is only applicable to models with convolutional architectures [2].

2.1 Saliency Maps

SM is one of the earliest and most widely used gradient-based explanation methods for deep neural networks in image classification. They were first introduced by Simonyan et al. in 2013 as a way to visualize which input pixels are most influential for a model's prediction [3].

The core idea of SM is to measure the sensitivity of a class prediction with respect to the input image. Pixels that cause a large change in the class score when slightly perturbed are considered important, while pixels with little influence are considered irrelevant. As a result, SM provide a pixel-level visualization that highlights regions of the image driving the classification decision [3].

Formally, given an input image $x \in \mathbb{R}^{H \times W \times C}$ and a neural network classifier $f(\cdot)$, the SM for class c is defined as:

$$S_c(x) = \left| \frac{\partial f_c(x)}{\partial x} \right| \tag{1}$$

In this equation:

- x denotes the input image, where H, W, and C represent the height, width, and number of channels (e.g., RGB) of the image, respectively.
- $f(x)$ represents the neural network model, which maps the input image to a set of class scores.

- $f_c(x)$ is the output score (logit) corresponding to class c. The logit is typically used instead of the softmax probability to avoid gradient saturation and to obtain more informative explanations.
- $\frac{\partial f_c(x)}{\partial x}$ is the gradient of the class score with respect to the input image. This gradient quantifies how sensitive the prediction for class c is to changes in each input pixel.
- $|\cdot|$ denotes the absolute value, which captures the magnitude of influence of each pixel regardless of whether it contributes positively or negatively to the class score.
- $S_c(x)$ is the resulting SM, where each pixel value reflects its importance for predicting class c.

In the context of image classification, the SM is typically visualized as a heatmap over the input image. High-intensity regions indicate pixels that have a strong influence on the model's decision, while low-intensity regions correspond to less important background information. This makes SMs a useful tool for interpreting and qualitatively evaluating the transparency of deep learning models.

$$S_c^{(i,j)}(x) = \max_{k \in \{1,\ldots,C\}} \left| \frac{\partial f_c(x)}{\partial x_{i,j,k}} \right| \tag{2}$$

where (i, j) denotes the spatial location of a pixel and k indexes the input channels. The resulting SM $S_c(x) \in \mathbb{R}^{H \times W}$ highlights the most influential regions of the image for predicting class c.

Overall, SMs (see Fig. 1) offer a simple yet effective way to visualize model attention and serve as a baseline method for explainability in image classification.

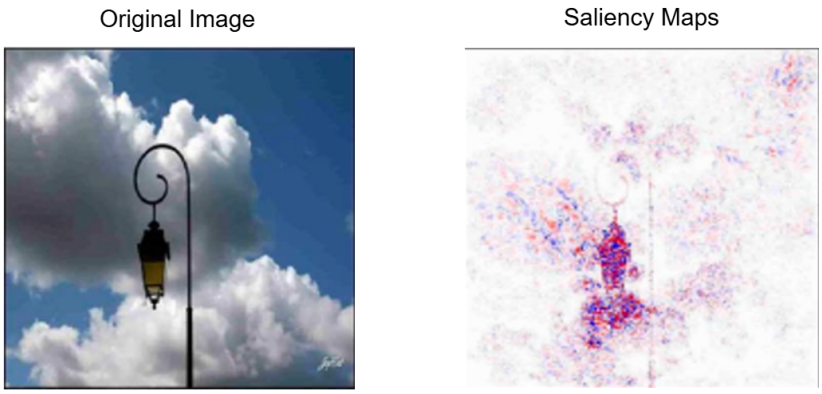

Figure 1: An example SM visualization

2.2 Integrated Gradients

IG is a gradient-based attribution method introduced by Sundararajan et al. in 2017 to address some of the limitations of standard saliency maps, such as gradient saturation and noise. The method provides a principled way to attribute a model's prediction to its input features while satisfying desirable axioms, including sensitivity and implementation invariance [4].

The key idea of IG is to measure the contribution of each input feature by integrating the gradients of the model output along a straight path from a baseline input to the actual input. The baseline represents the absence of information, such as a black image in the case of image classification [4]. By accumulating gradients along this path, IG captures both local and global effects of the input on the model's prediction [4].

Formally, given an input image $x \in \mathbb{R}^{H \times W \times C}$, a baseline image x', and a neural network classifier $f(\cdot)$, the IG attribution for class c is defined as:

$$\mathrm{IG}_c(x) = (x - x') \odot \int_{\alpha=0}^{1} \frac{\partial f_c\big(x' + \alpha(x - x')\big)}{\partial x}\, d\alpha \tag{3}$$

In this equation:

- x denotes the input image, where H, W, and C represent the height, width, and number of channels, respectively.
- x' is the baseline image, which represents a reference input with minimal or no information (e.g., a black or blurred image).
- $f(x)$ represents the neural network model that maps the input image to class scores.
- $f_c(x)$ is the output score (logit) corresponding to class c.
- $\alpha \in [0, 1]$ is a scaling parameter that defines points along the straight-line path from the baseline x' to the input x.
- $\frac{\partial f_c(\cdot)}{\partial x}$ is the gradient of the class score with respect to the input image, evaluated at points along the interpolation path.
- $\int_0^1 (\cdot)\, d\alpha$ denotes the path integral that accumulates gradients along the path from the baseline to the input.
- $\odot$ denotes element-wise multiplication, which scales the IG by the difference between the input and the baseline.
- $\mathrm{IG}_c(x)$ is the resulting attribution map, where each pixel value represents the contribution of that pixel to the prediction of class c.

In the context of image classification, IG produces a pixel-level attribution map that highlights image regions contributing most strongly to the model's decision (see Fig. 2). Compared to standard saliency maps, IG typically yields smoother and more stable explanations and is less affected by gradient saturation. These properties make it particularly suitable for quantitative evaluation against pixel-level ground-truth annotations.

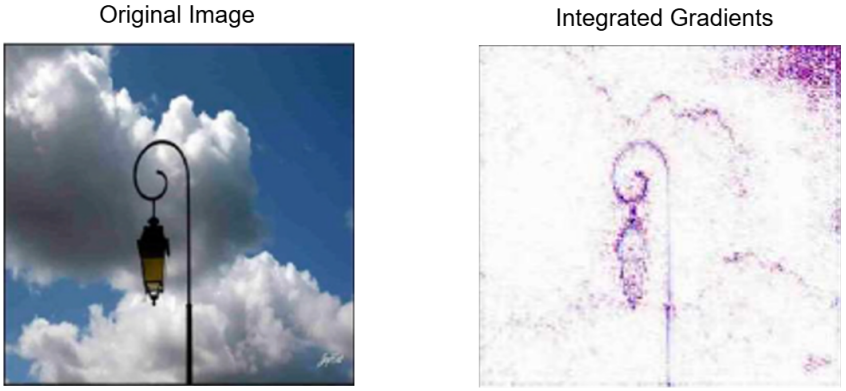

Figure 2: An example IG visualization

2.3 Gradient-weighted Class Activation Mapping

GC, first introduced in 2019 by Selvaraju et al., shares the same core idea as SMs: it uses derivatives to explain a model's specific decision for a specific input image. However, unlike SMs, GC computes gradients of the output only with respect to the last convolutional feature maps (i.e., up to the final

convolutional layer). Note that GC is a variant designed specifically for models with convolutional structures [2].

To compute GC [2], we first compute the neuron importance weights α_k^c using Eq. 4.

$$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial y^c}{\partial A_{ij}^k} \tag{4}$$

Where:

- α_k^c: Importance weight of feature map k for class c.
- c: Index of the target class of interest.
- k: Index of the feature map (channel) in the chosen convolutional layer.
- y^c: The score (logit, i.e., pre-softmax) for class c.
- A^k: The k-th feature map (activation map) of the chosen convolutional layer.
- A_{ij}^k: Activation value at spatial location (i, j) in feature map k.
- $\frac{\partial y^c}{\partial A_{ij}^k}$: Gradient of the class score w.r.t. that activation.
- i, j: Spatial indices over the height and width of the feature map.
- Z: Number of spatial locations in the feature map (typically $Z = H \times W$).

Finally, the GC heatmap [2] is computed by Eq. 5:

$$L_{\text{Grad-CAM}}^c = \text{ReLU}\left(\sum_k \alpha_k^c A^k\right) \tag{5}$$

Where:

- $L_{\text{Grad-CAM}}^c$: The GC localization map (heatmap) for the target class c.
- c: Index of the target class of interest.
- $\text{ReLU}(\cdot)$: Element-wise rectified linear unit, keeping only positive contributions to class c.
- $\sum_k$: Summation over all feature maps (channels) in the chosen convolutional layer.
- k: Index of a feature map (channel).
- α_k^c: Importance weight of feature map k for class c.
- A^k: The k-th feature map from the selected convolutional layer.

Suppose we use a pretrained EfficientNet-B0 [5] model with an input tensor of shape $(3, 224, 224)$. After passing through the final convolutional layer, the feature map tensor has shape $(1280, 7, 7)$ before entering the classifier head. Then, for a single input image, A has shape $(1280, 7, 7)$; the number of feature maps is $K = 1280$; and $Z = H \times W = 7 \times 7$. The weight vector α^c has shape $(1280,)$, and the GC map $L_{\text{Grad-CAM}}^c$ has shape $(7, 7)$.

GC's final heatmap has a much smaller spatial resolution than the original input image. Therefore, for visualization purposes, bilinear interpolation is used to upsample it back to the original input size.

As suggested in the original GC paper, the feature maps from the last convolutional layer should be used because they provide the best compromise between high-level semantics and detailed spatial information [2]. GC does not require any hyperparameters.

Figure 3 shows an example of GC. From left to right, the first image is the original input, the second is the original GC heatmap, the third is the GC heatmap upsampled to match the input resolution, and the fourth is an overlay of the input image and the GC heatmap to visualize how the highlighted pixels correspond to the image content.

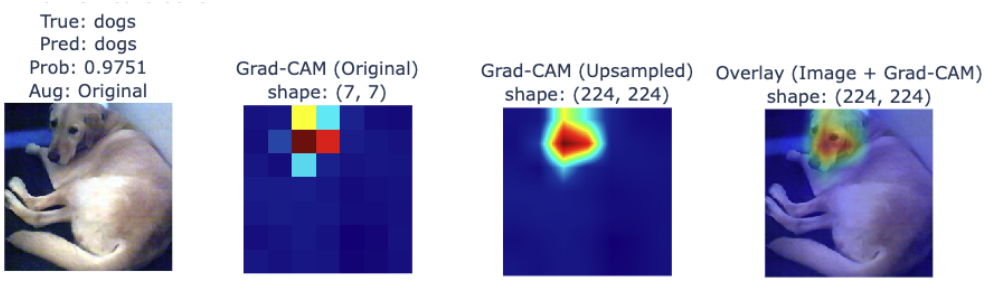

Figure 3: An example GC visualization

3 Experiments

In this section, we describe the training pipeline of our model as well as the methodology used to compare different explainable AI methods.

3.1 Dataset

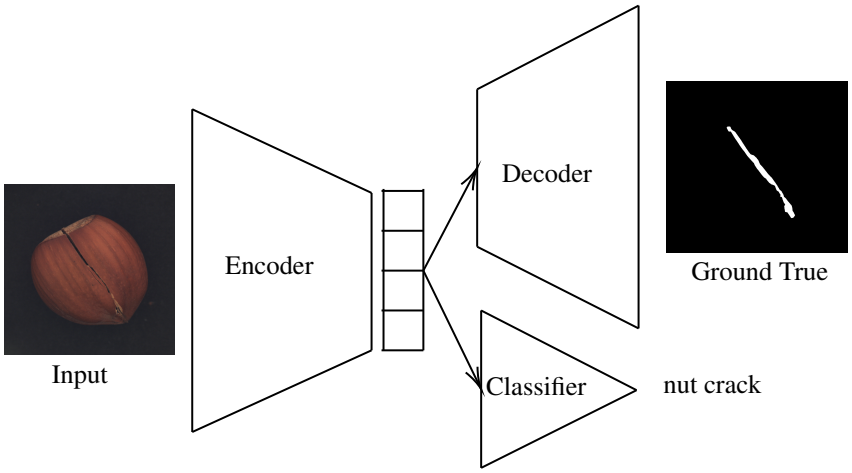

Figure 4: Training Pipeline

We conduct our experiments on a small subset of MVTec AD dataset [6]. Specifically, we focus on images of hazelnuts. Each input image is associated with two types of annotations:

- an image-level label indicating whether the hazelnut is defective (e.g., nut crack, as shown in Fig. 4 left), and
- a pixel-level ground-truth mask that localizes the defective regions. The pixel-level annotation is binary, where black pixels represent the background and white pixels indicate defective areas (as shown in Fig. 4 right).

In this classification task, we consider a total of five classes, namely: crack, cut, good, hole, and print.

3.2 Training Pipeline

Our model follows a multitask learning framework based on an autoencoder architecture with an additional classification head. The autoencoder is trained to reconstruct the input image as accurately as possible, while the classifier simultaneously predicts the defect label of the image. This joint learning setup enables the model to capture both low-level pixel information and high-level semantic features relevant for defect classification (see Fig. 4).

The training objective is defined as a weighted sum of a classification loss and a reconstruction loss:

$$\mathcal{L} = \mathcal{L}_{\mathrm{CE}}(y, \hat{y}) + \beta \mathcal{L}_{\mathrm{BCE}}(y_{\mathrm{gt}}, y_{\mathrm{res}}), \tag{6}$$

where:

- $\mathcal{L}_{\mathrm{CE}}$ denotes the cross-entropy loss used for the image-level classification task.
- y is the ground-truth image-level defect label.
- $\hat{y}$ is the predicted defect label produced by the classifier.
- $\mathcal{L}_{\mathrm{BCE}}$ denotes the binary cross-entropy loss used to measure the reconstruction quality at the pixel level.
- y_{gt} represents the ground-truth pixel-level defect mask.
- y_{res} denotes the reconstructed output image produced by the decoder.
- β is the trade-off hyperparameter between classification and reconstruction.

By jointly optimizing these two loss components, the model learns representations that are effective for defect classification while preserving spatial information that is crucial for pixel-level explanations.

We train a model with an autoencoder-based architecture combined with a classification head. The encoder follows the structure of EfficientNet-B0, while the decoder is composed solely of convolutional building blocks, including ConvTranspose2D, ReLU, Batch Normalization, and Conv2D layers. The classification head is a linear layer, which is attached to the encoder output to perform image-level classification.

Pretrained weights of EfficientNet-B0 are used to initialize the encoder. During training, all model parameters are optimized jointly, and no layers are frozen. This allows the network to fully adapt both the feature extractor and the downstream components to the target dataset.

The model is trained using PyTorch on an NVIDIA RTX A5000 GPU. The source code is publicly available on GitHub at: `https://github.com/khoaphamanh/project_iml`. Reimplementing the code and reproducing the experimental results requires approximately 30 minutes on a standard machine with GPU support.

All experiments are conducted using 10 fixed random seeds, specifically: 42, 43, 44, 45, 46, 47, 48, 49, 50, and 51.

The hyperparameters used during training are summarized in Tab. 1.

Table 1: Training hyperparameters used for model optimization

Hyperparameter	**Value**
Number of epochs	30
Batch size	32
Learning rate	0.001
Number of workers	4
Consistency weight (β)	1
Base number of decoder channels	16

The performance of the model across 10 different random seeds is stable summarized as follows in Tab. 2 and Fig. 5.

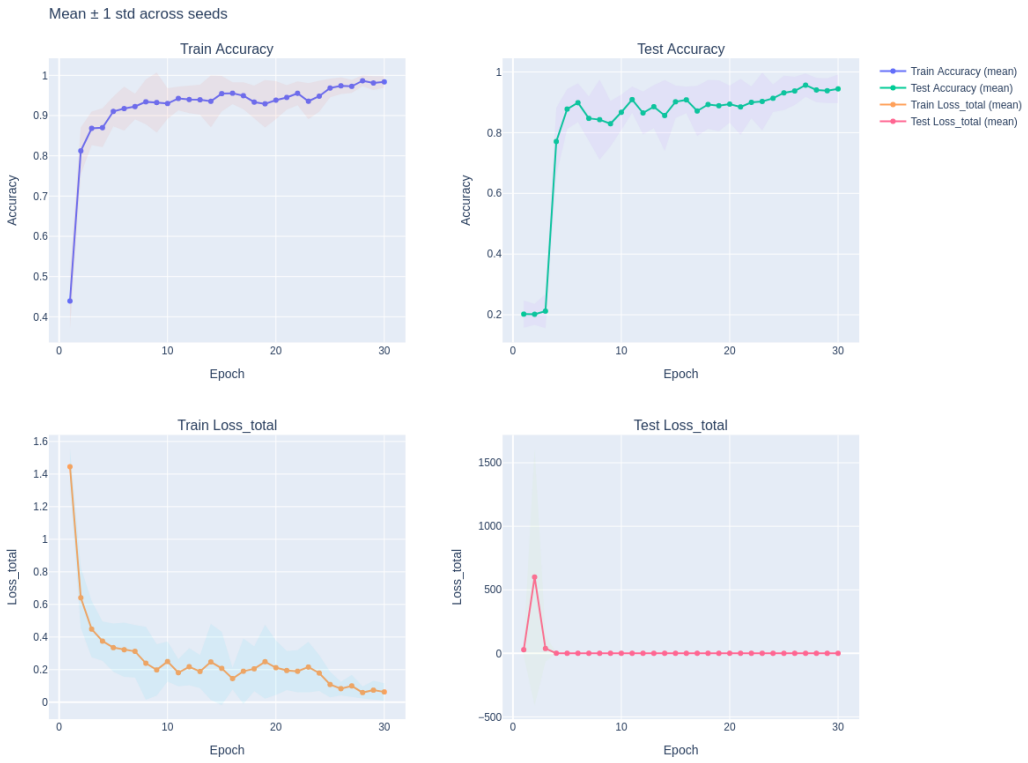

Figure 5: Loss and accuracy of the model averaged over 10 random seeds

Table 2: Model performance averaged over 10 random seeds (Epoch 30)

Split	Loss Total	Loss CE	Loss BCE	Balanced Accuracy
Train	0.0631 ± 0.0530	0.0581 ± 0.0527	0.0050 ± 0.0007	0.9837 ± 0.0144
Test	0.2087 ± 0.1745	0.2006 ± 0.1736	0.0081 ± 0.0014	0.9443 ± 0.0467

3.3 Evaluation of Explanation Faithfulness via Top-K Overlap

To quantitatively evaluate the faithfulness of the explanation methods GC, SM and IG, we adopt a Top-K Overlap metric based on pixel-level ground-truth annotations.

The evaluation procedure is defined as follows:

1. We define K as the number of positive (white) pixels in the ground-truth defect mask, i.e., the number of pixels labeled as defective.

2. For each explanation method, we select the Top-K pixels with the highest attribution scores from the corresponding explanation map (see Fig. 6 row 3). These pixels represent the most important regions identified by the method.

3. We then compute the number of selected Top-K pixels that fall within the ground-truth defect region (i.e., pixels labeled as white in the mask, see Fig. 6 row 4).

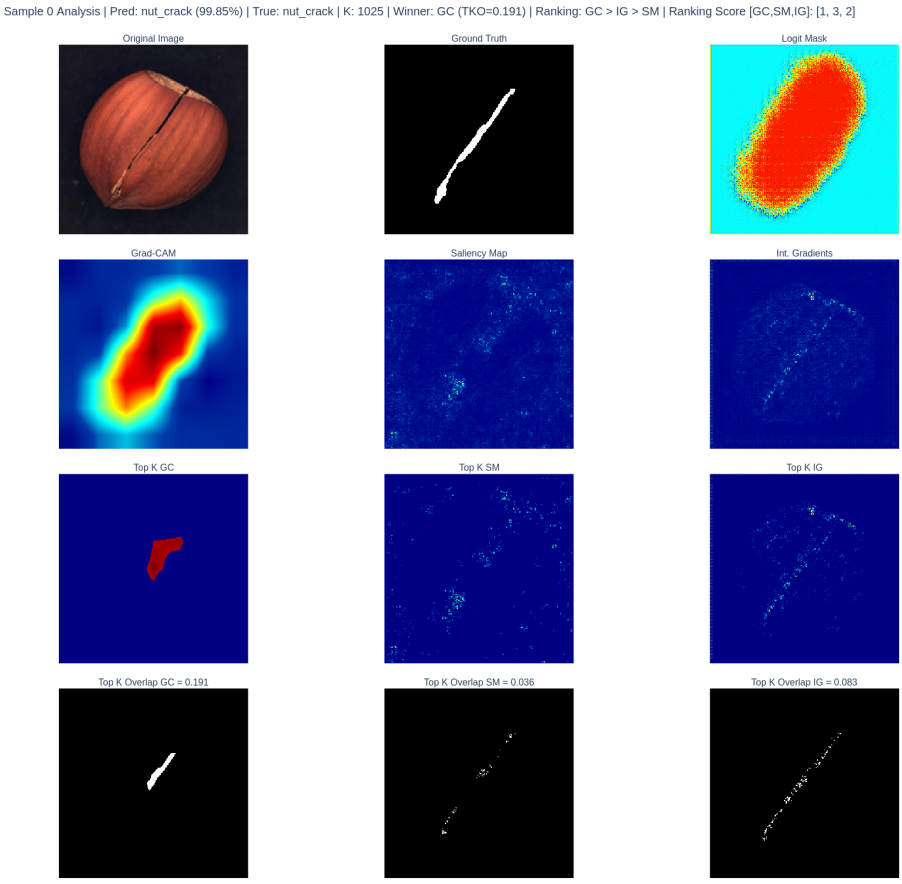

Figure 6: Top-K Overlap for GC, SM and IG from nut crack image

3.4 Quantitative Results and Win-Rate Analysis

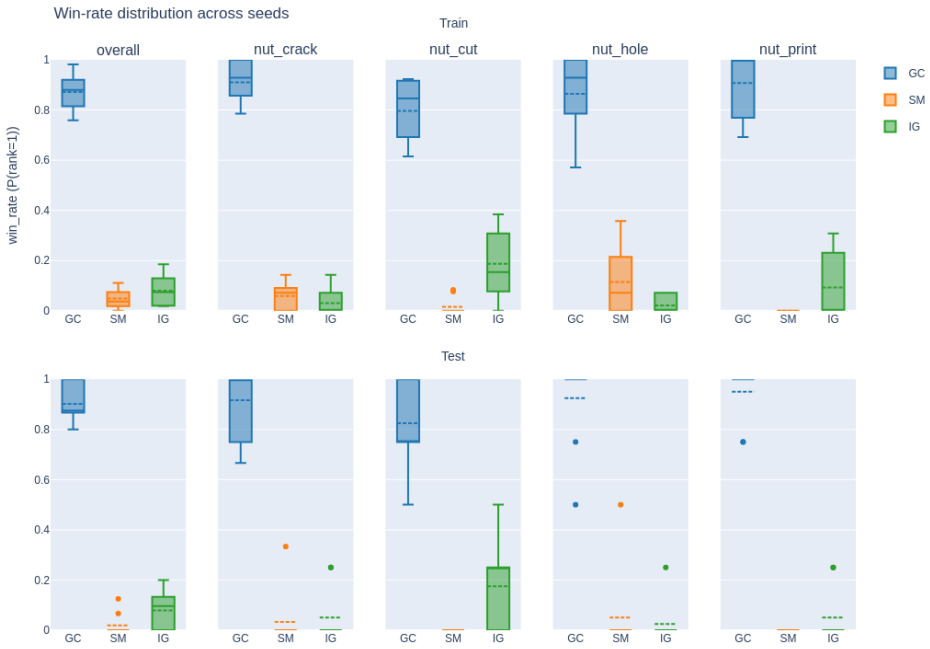

Figure 7: Results as win-rate averaged over 10 random seeds

As shown in Fig. 6, GC achieves a Top-K Overlap score of 19.1%, meaning that 19.1% of the highest-scoring pixels identified by GC fall within the ground-truth defect region. This is substantially higher than SM, which achieves 3.6%, and IG, which achieves 8.3%. Based on this metric, we consider GC to be the winner for this particular example.

To obtain a more comprehensive evaluation, we compute the win-rate of each explanation method across the entire training and test datasets using 10 different random seeds. For each image, a method is considered the winner if it achieves the highest Top-K Overlap score among the three methods. Importantly, we compute the Top-K Overlap only on images that are correctly classified by the model, ensuring that explanations are evaluated only when the prediction itself is reliable.

Table 3: Win-rate on the training set averaged over 10 random seeds

Class	GC	SM	IG
Overall	**0.8713** $\pm$ 0.0678	0.0486 $\pm$ 0.0382	0.0801 $\pm$ 0.0564
Crack	**0.9104** $\pm$ 0.0740	0.0591 $\pm$ 0.0543	0.0305 $\pm$ 0.0495
Cut	**0.7968** $\pm$ 0.1119	0.0160 $\pm$ 0.0321	0.1872 $\pm$ 0.1181
Hole	**0.8643** $\pm$ 0.1445	0.1143 $\pm$ 0.1204	0.0214 $\pm$ 0.0327
Print	**0.9077** $\pm$ 0.1278	0.0000 $\pm$ 0.0000	0.0923 $\pm$ 0.1278

Table 4: Win-rate on the test set averaged over 10 random seeds

Class	GC	SM	IG
Overall	**0.9017** $\pm$ 0.0739	0.0192 $\pm$ 0.0405	0.0792 $\pm$ 0.0712
Crack	**0.9167** $\pm$ 0.1291	0.0333 $\pm$ 0.1000	0.0500 $\pm$ 0.1000
Cut	**0.8250** $\pm$ 0.1601	0.0000 $\pm$ 0.0000	0.1750 $\pm$ 0.1601
Hole	**0.9250** $\pm$ 0.1601	0.0500 $\pm$ 0.1500	0.0250 $\pm$ 0.0750
Print	**0.9500** $\pm$ 0.1000	0.0000 $\pm$ 0.0000	0.0500 $\pm$ 0.1000

Fig.7 shows the distribution of win-rates across seeds. Under the condition that the classification model achieves strong predictive performance, GC consistently dominates the comparison. Its win-rate is never below 79% and reaches up to 95%, both on the training and test sets, across the entire dataset as well as for each individual defect class. IG ranks second, while SM consistently performs worst (see Tab. 3 and 4).

The inferior performance of SM and IG in terms of Top-K Overlap can be explained by the nature of their attribution mechanisms. Both methods compute gradients of the output with respect to the input pixels, thus primarily capturing local sensitivity rather than structured semantic importance. As a result, their attribution maps tend to be noisy or fragmented, especially in scenarios where the model's prediction is highly confident.

In contrast, GC computes gradients of the output with respect to the final convolutional feature maps. Because these feature maps encode high-level semantic representations and object-level spatial structure, Grad-CAM is better able to localize coherent regions corresponding to the defect. This object-level reasoning leads to higher attribution scores within the crack region and consequently a substantially higher Top-K Overlap score.

4 Limitations, Future Work & Conclusion

4.1 Limitations

From a qualitative perspective, GC consistently produces smooth and visually coherent heatmaps with minimal noise. However, this method is inherently designed for CNNs and assumes image-based inputs. As a result, GCM cannot be applied to other data modalities or neural network architectures, such as transformer-based models or non-image inputs.

In contrast, gradient-based methods such as SM and IG are more generally applicable, as they rely solely on gradients with respect to the input. This allows them to be used across a wider range of data types and model architectures. Nevertheless, this generality often comes at the cost of increased noise and reduced visual interpretability.

4.2 Future Work

The smoothness and noise-free nature of GC representations constitute one of its main strengths. Motivated by this property, a promising direction for future research is to explore the application of GC in more complex vision tasks, such as object detection in video data. Extending GC to spatiotemporal settings may provide deeper insights into model behavior across both spatial and temporal dimensions.

4.3 Conclusion

In the context of image classification using well-performing convolutional neural networks, GC emerges as the most suitable explanation method. It consistently produces smooth, low-noise heatmaps and demonstrates superior qualitative interpretability compared to the other methods considered. Among the three approaches, GC is also the most recent technique.

IG partially addresses the noise issue observed in standard SM and generally provides more stable attributions. However, it requires significantly higher computational resources due to the integration over multiple interpolation steps. SM, while computationally efficient and easy to implement, suffer from substantial noise and yield less informative explanations. Based on our empirical evaluation, SM represent the least effective method in terms of interpretability quality.

Overall, this project highlights that the choice of explanation method must be carefully aligned with the task, input modality, and model architecture. For end users and stakeholders, selecting an appropriate interpretability technique based on relevant benchmarking criteria is essential to ensure meaningful and trustworthy model explanations.

References

[1] M. Lindauer, "Interpretabel machine learning lecture."

[2] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 618–626.

[3] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *arXiv preprint arXiv:1312.6034*, 2013.

[4] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," in *International conference on machine learning*. PMLR, 2017, pp. 3319–3328.

[5] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*. PMLR, 2019, pp. 6105–6114.

[6] P. Bergmann, M. Fauser, D. Sattlegger, and C. Steger, "The mvtec anomaly detection dataset: A comprehensive real-world dataset for unsupervised anomaly detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.