

# Grad-CAM Explanation

November 8, 2024

ⓘ This conversation may reflect the link creator's personalized data, which isn't shared and can meaningfully change how the model responds.



Explain Grad-CAM in short. Explain computing gradients, and weights obtained from global average pooling in detail with mathematical equations.



Grad-CAM, or Gradient-weighted Class Activation Mapping, is a technique used to visualize which regions of an input image contribute most to a deep neural network's decision. It does this by utilizing the gradients of a specific target class flowing into the last convolutional layer to produce a coarse localization map highlighting the important regions in the image for that class.

Here's a breakdown of how Grad-CAM works, focusing on gradients and the weights computed from global average pooling.

## 1. Computing Gradients

To generate the Grad-CAM, we first compute the gradient of the score for the target class  $y^c$  with respect to the feature maps  $A^k$  of a convolutional layer. Here:

- $y^c$  is the score (logits before softmax) of the target class  $c$ .
- $A^k$  is the  $k$ -th feature map in the last convolutional layer.

The gradient is computed as:

$$\frac{\partial y^c}{\partial A^k}$$

This gradient indicates how important each pixel in the feature map  $A^k$  is for the class  $c$ . It shows how the score for class  $c$  changes with respect to changes in  $A^k$ .

## 2. Weights from Global Average Pooling (GAP)

To summarize the importance of each feature map  $A^k$  for the target class, Grad-CAM uses global average pooling on the gradients to obtain a weight for each feature map. These weights, denoted as  $\alpha_k^c$ , represent the importance of feature map  $A^k$  for the target class  $c$ .

The weight  $\alpha_k^c$  is calculated as:

$$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial y^c}{\partial A_{i,j}^k}$$

where:

- $A_{i,j}^k$  represents the activation at position  $(i, j)$  in feature map  $A^k$ .
- $Z$  is the total number of pixels in each feature map (i.e.,  $Z = \text{height} \times \text{width}$ ).

This global average pooling gives a single weight  $\alpha_k^c$  for each feature map, indicating how much the activation map  $A^k$  contributes to the target class  $c$ .

### 3. Generating the Grad-CAM Heatmap

Once we have the weights  $\alpha_k^c$ , we can generate the Grad-CAM heatmap by creating a weighted combination of the feature maps  $A^k$  and passing it through a ReLU activation to keep only positive values. The heatmap  $L^c$  for class  $c$  is given by:

$$L^c = \text{ReLU} \left( \sum_k \alpha_k^c A^k \right)$$

This heatmap  $L^c$  highlights the regions in the input image that are most important for the target class prediction. Applying ReLU ensures that only the positive contributions (indicating evidence for the class) are considered.

### Summary

1. **Compute gradients**  $\frac{\partial y^c}{\partial A^k}$  to determine the influence of each feature map on the target class.
2. **Obtain weights**  $\alpha_k^c$  by applying global average pooling to the gradients.
3. **Generate the heatmap**  $L^c = \text{ReLU} \left( \sum_k \alpha_k^c A^k \right)$  to visualize the important regions for the target class.

This approach makes Grad-CAM a powerful and interpretable visualization technique for convolutional neural networks.



Explain how to visualize by superimposing the heatmap.



To visualize a Grad-CAM heatmap by superimposing it on the original image, we blend the Grad-CAM output (the heatmap) with the original image to highlight important regions identified by the model. Here's a step-by-step guide on how to create this visualization:

### 1. Generate the Grad-CAM Heatmap

Following the Grad-CAM process:

- Compute the heatmap  $L^c$  as described previously, which represents important regions for the target class. This heatmap will have positive values where the model has focused attention for class  $c$ .

- Apply a ReLU activation to the final weighted combination of feature maps, ensuring the heatmap only has non-negative values.

## 2. Normalize the Heatmap

For better visualization, the heatmap values should be normalized to a range of  $[0, 1]$ :

$$L_{\text{norm}}^c = \frac{L^c - \min(L^c)}{\max(L^c) - \min(L^c)}$$

This normalization scales the heatmap values, making the most important regions of the image brighter in the visualization.

## 3. Resize the Heatmap to Match the Input Image Size

Since the heatmap  $L_{\text{norm}}^c$  comes from a convolutional layer, it typically has a smaller spatial resolution than the original input image. Resize the heatmap to match the dimensions of the input image  $H \times W$  using an interpolation method (e.g., bilinear interpolation) to maintain smoothness.

## 4. Apply a Color Map to the Heatmap

To make the heatmap visually informative, apply a color map (e.g., ``jet``, ``hot``, or ``viridis``) to the normalized heatmap. This color mapping converts the grayscale heatmap into a color image where colors represent the intensity of the activation:

- For example, using a ``jet`` colormap:
  - Low values are mapped to blue,
  - Medium values to green/yellow,
  - High values to red.

Let's denote this color-mapped heatmap as  $H_{\text{colored}}$ .

## 5. Superimpose the Heatmap on the Original Image

Combine  $H_{\text{colored}}$  with the original image  $I$  by overlaying it with some transparency. The transparency controls the balance between the original image and the heatmap.

Let  $\alpha$  (usually between 0.3 and 0.5) be the transparency factor. The final superimposed image  $I_{\text{superimposed}}$  is given by:

$$I_{\text{superimposed}} = \alpha \cdot H_{\text{colored}} + (1 - \alpha) \cdot I$$

This superimposed image combines the salient regions (from  $H_{\text{colored}}$ ) with the original content, making it easy to see which parts of the image contributed most to the model's prediction.

## 6. Display or Save the Result

Once you have  $I_{\text{superimposed}}$ , you can display it or save it using a visualization library like Matplotlib, OpenCV, or PIL in Python.

## Example Code

Here's a simple Python example to visualize the Grad-CAM output using OpenCV and Matplotlib:

```
python

import cv2
import numpy as np
import matplotlib.pyplot as plt

# Assume `heatmap` is the normalized Grad-CAM heatmap and `original_image` is the input image

# Step 1: Resize heatmap to match original image size
heatmap_resized = cv2.resize(heatmap, (original_image.shape[1], original_image.shape[0]))

# Step 2: Apply color map to heatmap
heatmap_colored = cv2.applyColorMap(np.uint8(255 * heatmap_resized), cv2.COLORMAP_JET)

# Step 3: Superimpose heatmap on the original image with transparency
alpha = 0.4
superimposed_image = cv2.addWeighted(heatmap_colored, alpha, original_image, 1 - alpha, 0)

# Step 4: Display the superimposed image
plt.imshow(cv2.cvtColor(superimposed_image, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

This visual representation helps in understanding which regions the model considered crucial for the prediction, making it a powerful tool for interpretability.



Explain where we can apply these visualization in real life tasks (like model debugging, improving model design, explaining model decisions in applications such as medical diagnostics). Answer very briefly and in the context of the concept "Explainable AI".



Grad-CAM visualizations play a crucial role in Explainable AI by making model decisions interpretable and trustworthy. Here are key real-life applications:

1. **Model Debugging:** Grad-CAM helps identify if a model is focusing on irrelevant regions, allowing developers to fine-tune the model, improve dataset quality, or adjust model design for better performance.
2. **Improving Model Design:** By visualizing feature importance, Grad-CAM reveals whether the architecture is extracting meaningful patterns, guiding designers in refining layers or feature extraction techniques.
3. **Medical Diagnostics:** In fields like radiology and pathology, Grad-CAM highlights areas of focus in medical images (e.g., tumors or abnormalities), making AI decisions more transparent to healthcare professionals, building trust, and supporting diagnosis.
4. **Autonomous Driving:** In driver-assistance systems, Grad-CAM can show which objects or regions (like pedestrians or traffic signs) influenced the model's decisions, aiding developers in validating safety-critical AI.

5. **Legal and Financial Services:** Explainability in decision-making models is vital for regulatory compliance. Grad-CAM aids in identifying biases or misaligned reasoning, especially in credit scoring or fraud detection.

In each of these areas, Grad-CAM's visualizations help bridge the gap between complex model workings and human understanding, enhancing transparency, trust, and accountability.