Gradient-based explainability methods

Explainable and Trustworthy AI

Eliana Pastor

Stages of Explainability

- Explainability involves the entire AI development pipeline

Pre-modelling explainability

Explainable modeling

Post-modelling explainability

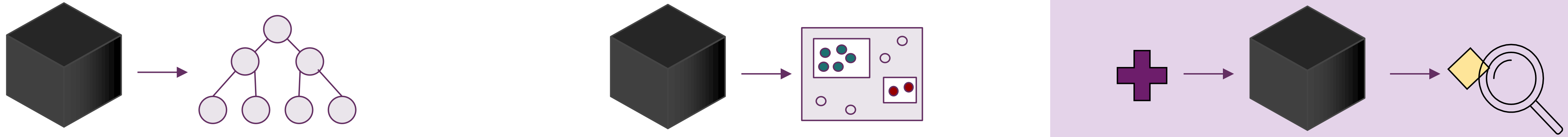
Before building the model
- Data exploration
- Data selection
- Feature engineering

Build inherently interpretable models
- Manage the accuracy and interpretability trade-off

After model development
- Explaining predictions and behavior of trained models

Scope of Explainability

- *What do we explain?*



Global

Subgroup

Individual/local

How the model globally works

How the model behaves in data subgroups

Explaining the reasons behind individual predictions

Gradient-based explainability methods

- Techniques that leverage the **gradient** information **of the model with respect to the input features** to identify which features are most influential in the model's decision-making process.

- Gradient-based methods differ in how the gradient is computed

- Generally, the explanation has the same size as the input
- They assign each part of the input a value that is interpreted as the relevance
  - e.g., for images, importance for each pixel of the image
  - e.g., for text, importance of each token

Saliency map

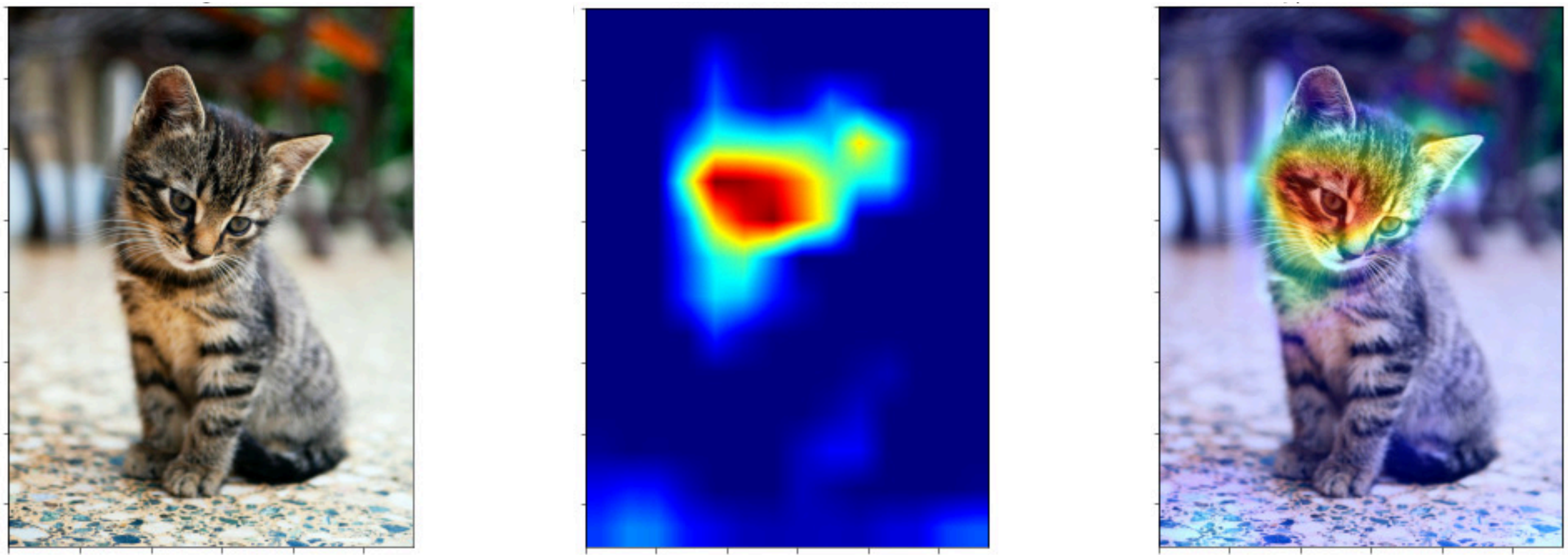


Image from: https://dlhr.de/8

Saliency map

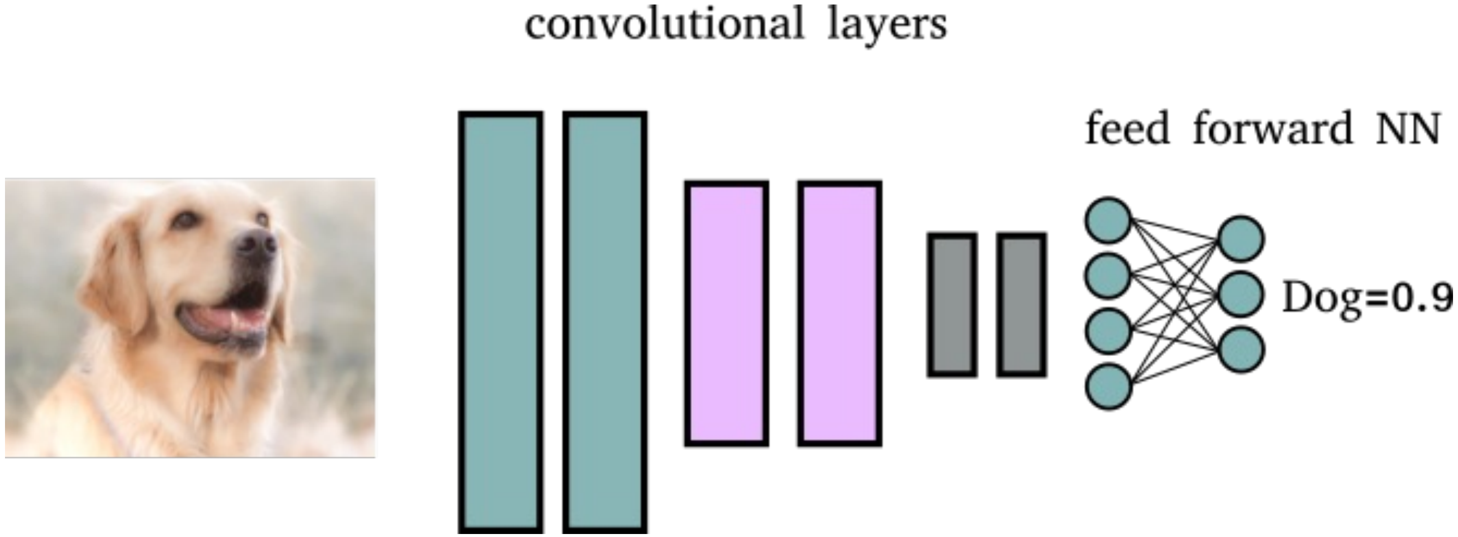
- **Saliency** is a **visualization** technique to highlight the important regions or features in an input data sample
    - It is a way to visualize **feature attributions**, i.e., to represent the attribution scores assigned to each input feature
- Brighter regions in the saliency map indicate higher saliency or importance, highlighting the regions of interest in the input data

I am really happy

Vanilla Gradient

Compute the gradient of the loss function with respect to the input

- Proposed for image data → respect the input pixel



Gradients: from backprogation to saliency

Gradient in the training process.

- During the training process, gradients are computed with respect to the model parameters to update them.
    - Goal is to adjust model parameters in a direction that minimizes the loss function. This is typically done using techniques like backpropagation.

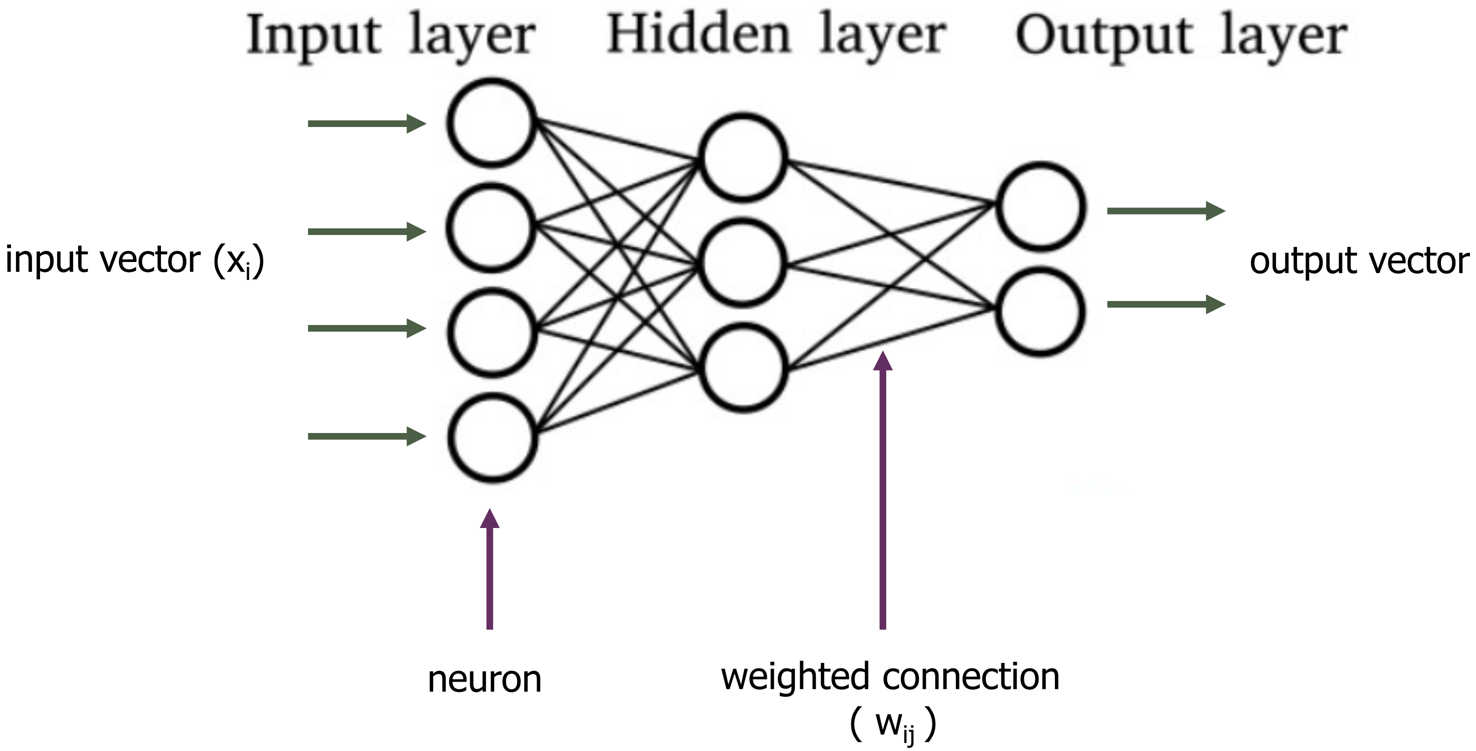
Gradients in explainability.

- In gradient-based explainability methods, instead of computing gradients with respect to the parameters, **we compute gradients with respect to the input features themselves**.
    - We analyze how changes in the input features affect the output directly.

Gradients – at training time

Backpropagation

$$\frac{\partial L}{\partial w}$$



input vector (x_i)

output vector

neuron

weighted connection
(w_{ij})

Gradients for explainability

Given.

- A network trained for C classes. Output of the network for input I is a prediction vector $\mathrm{F}(I) = [F_1(I), \ldots, F_C(I)]$
- For a class c, $F_c(I)$ is the score of the class c

Goal.

- Given an input $\mathrm{I_o}$with p features and a class c, we want to compute a relevance score for each features for class c

$$R^c = \left[R_1^c, . . , R_p^c\right]$$

for the score F_c

For image data, the features can be the pixels of the image with width w and height h, with $\mathrm{p} = w \times h$

Gradients for explainability

$F_c(I)$ is the score of the class c and R^c relevance score for each features for class c

How.

We derive R^c by computing the gradient of class score of interest with respect to the input pixels:

$$\left.\frac{\partial F_c}{\partial I}\right|_{I_0}$$

i.e., the derivative of F_c with respect to the input for the given image $\mathrm{I_o}$

Gradients for explainability

The idea is to model the score model F_c as a linear function. Since F_c is non-linear, we approximate with the first-order Taylor expansion

$$F_c(I) \approx w^T \cdot \ I \ + \ b = R_c^T \cdot \ I \ + \ b$$

Where the weight vector $w = R_c$ is t derivate of the score and b is the bias of the model.

The weights R_c define the importance of the feature of I for the class c.

Gradients for explainability

- Let F be the model function and $F_c(x)$ the score function for the input x for the class c

- We compute the input gradient importance

$$\nabla_x F_c(x) = [\frac{\partial F_c}{\partial x_1}, \dots, \frac{\partial F_c}{\partial x_p}]$$

where $\nabla_x F_c(x)$ denotes the gradient of $F_c(x)$ for input x and $\frac{\partial F_c}{\partial x_i}$ is the partial derivative of $F_c(x)$ with respect to the i-th input feature.

We typically compute it via backpropagation.

Gradients for explainability

Interpretation of the image-specific class saliency

- Features with **larger gradients** indicate that **small changes in those features will result in more significant changes** in the model's output
  - Magnitude of the derivative indicates which features need to be changed the least to affect the class score the most
  - Gradients tell us which features (e.g., pixels) have the steepest local relative to your model's prediction at a given point along your model's prediction function

From gradient to saliency map

- $F_c(I)$ = score for class c, e.g., logits

- w has size as the input
- For images, $I_0 \in \mathbb{R}^{H \times W \times K}$ (e.g., K=3 for RGB) -→ $w \in \mathbb{R}^{H \times W \times K}$

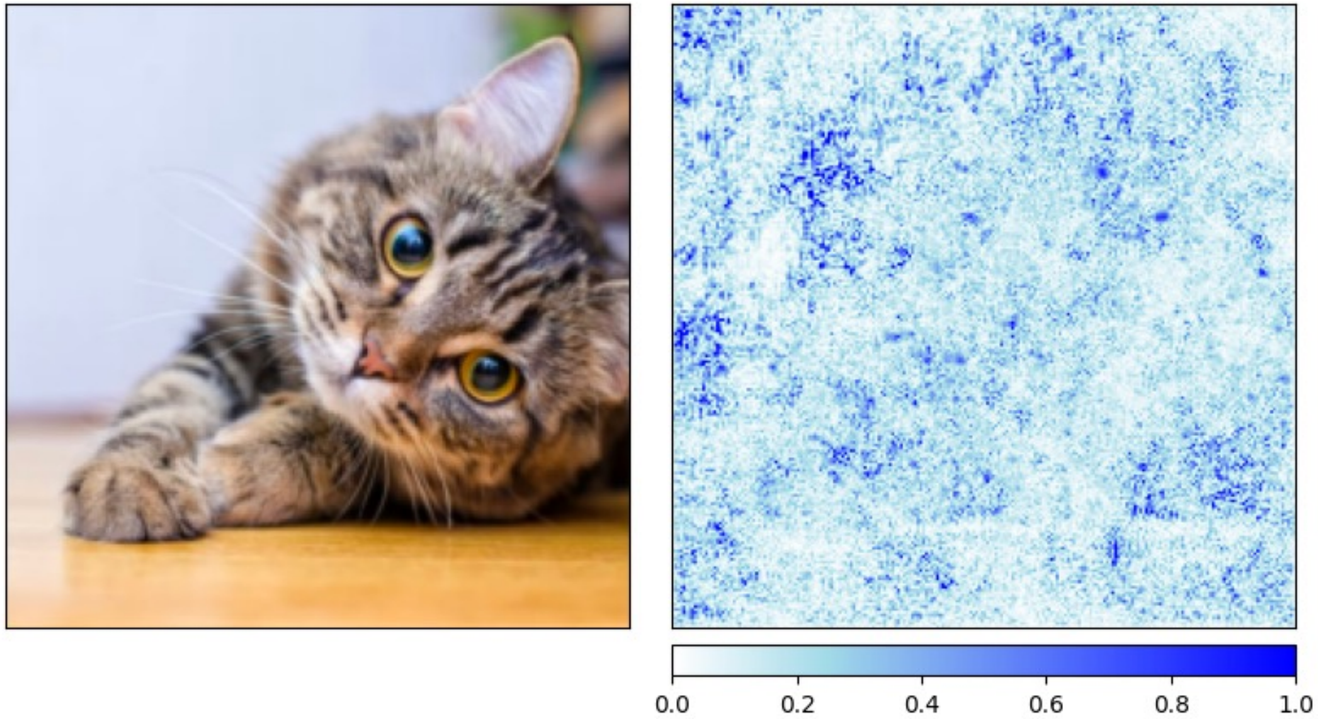
- If we want to plot a saliency map over the image, we need to aggregate the scores w to have a saliency map of size $H \times W$
- We can derive the saliency map $\mathrm{M} \in \mathbb{R}^{H \times W}$ as follows

$$M_{ij} = \max_k |w_{ijk}|$$

i.e., we collapse the challenge dimensions

Vanilla gradient – Limitation

- Noisy!
  - Derivative can fluctuate great at small scales
  - Slight variations to the input data can result in significant changes in the model output (thus in the gradients), resulting in noisy gradients and instability



SmoothGrad

- Vanilla Gradient can produce noisy saliency maps, hence difficult to interpret

- SmothGrad averages the gradients for ‘noisy inputs’

$$\frac{1}{N}\sum_{1}^{N}\nabla_x F_c(x+\epsilon)$$

Where ϵ is Gaussian Noise.

We create a **smoothing effect**. The intuition is that by averaging the gradients over modifications of the input, smooths out fluctuations & average out noise

SmoothGrad

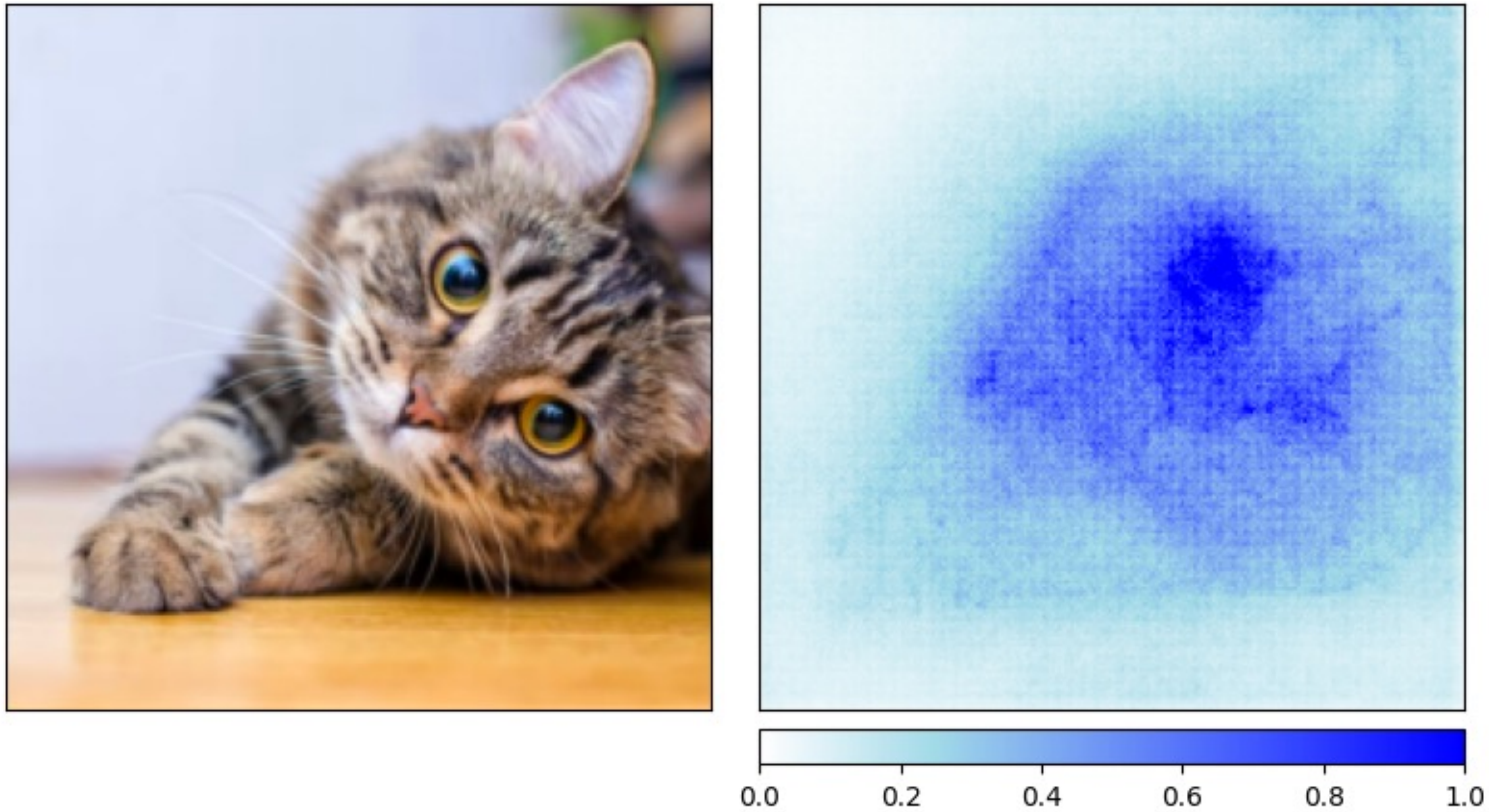
Process.

- Generate N versions of the input by adding noise to it.
- Compute the gradients for the N input, thus generating N relevance scores R_c
- Average the pixel attribution maps

- Parameters
  - Noise level
  - Number of samples

- Note that we can generally apply a generic gradient-based explainability approach

Smooth-Grad + Vanilla Gradient - Example



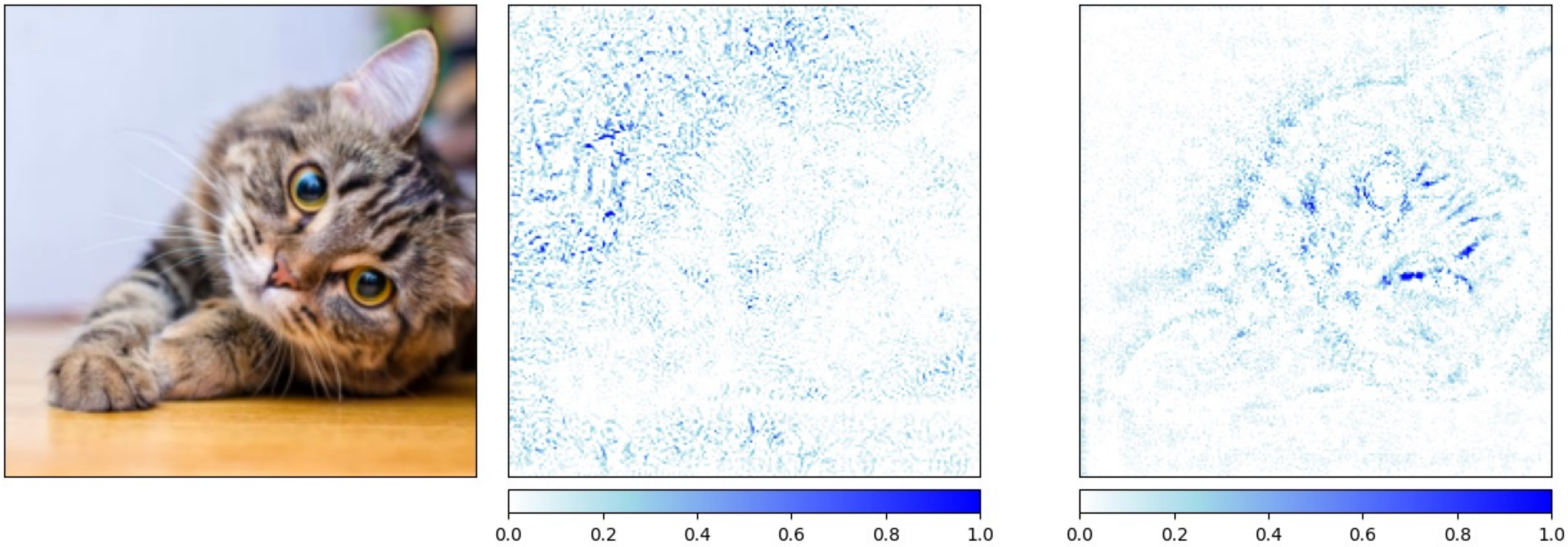
Gradient x Input

- Small variation of Vanilla Gradient
- The gradients w.r.t. the input are multiplied for the input (element wise product)

$$\nabla_x F_c(x) \odot x$$

- It generally provides better results

Gradient x Input - Example

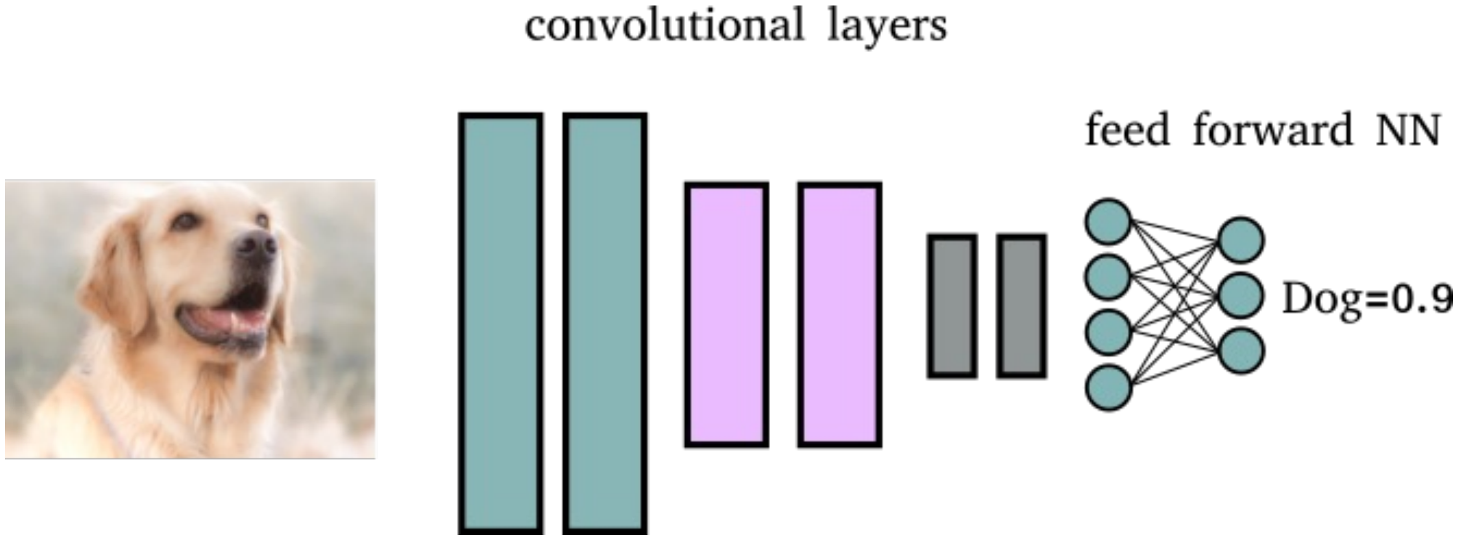


Gradient x Input

Gradient x Input + SmoothGrad

GradCAM

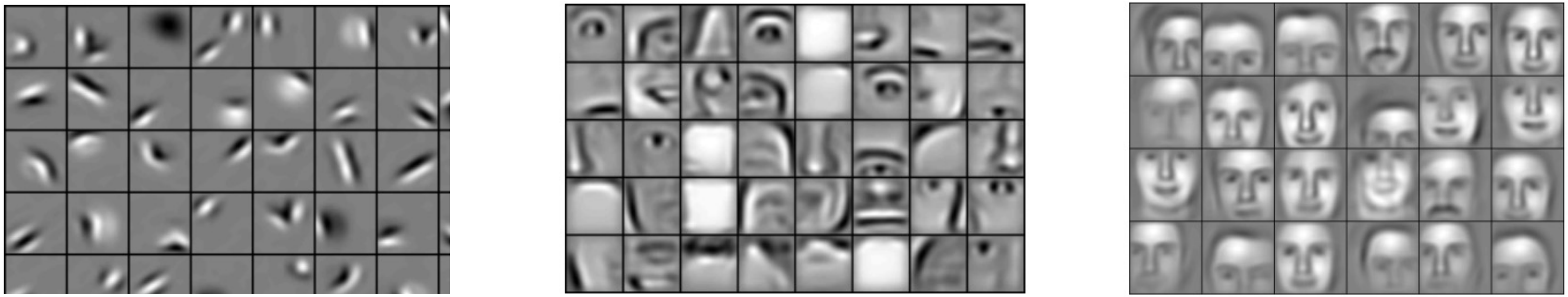
- Gradient-weighted Class Activation Mapping
- Suitable for CNN-based architectures



GradCAM

Intuition.

- Deeper representations in a CNN capture **higher-level visual constructs**



Lee et al, “Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations”, ICML 2009.

GradCAM

Intuition.

- Deeper representations in a CNN capture **higher-level visual constructs**
- Convolutional layers naturally retain spatial information which is lost in fully-connected layers, so we can expect the **last convolutional layers** to have the best compromise between **high-level semantics** and **detailed spatial information**.

- Grad-CAM uses the **gradient** information flowing into the **last convolutional layer** of the CNN to assign importance values to each neuron for a decision.
  - Explain activations in the output layer decisions

GradCAM

- Difference→ the gradient is not backpropagated back to the image
- Grad-CAM → Propagated (usually) to the last convolutional layer

$$\cancel{\left.\frac{\partial F_c}{\partial I}\right|_{I_0}}$$

- Let $A_k \in \mathbb{R}^{U \times V}$ be feature map activations of a convolutional layer (typically the last). The layer produces K feature maps
  - Each element is by i, j. Hence, $A_{i,j}^{k}$ refers to the activation at location (i, j) of the feature map A_k
- Grad-CAM produces a coarse localization map that highlights important regions of the image, i.e., **Layer-wise importance**, $Layer\text{ -}R^c \in \mathbb{R}^{U \times V}$ for a class c, where U, V is the shape of A_k

$$\frac{\partial F_c}{\partial A_{i,j}^{k}}$$

GradCAM - Process

a. Compute the gradient of the score for class c, F_c, with respect to feature map activations A_k of that convolutional layer of interest (usually the last) -- via backpropagation

$$\frac{\partial F_c}{\partial A^k}$$

b. Compute the average for each output channel – global average pooling

$$\alpha_{\mathrm{k}}^{\mathrm{c}} = \frac{1}{\mathrm{Z}} \sum_i \sum_j \frac{\partial F_c}{\partial A_{i,j}^k}$$

$\alpha_{\mathrm{k}}^{\mathrm{c}}$ captures the *importance* of feature map k for a class c

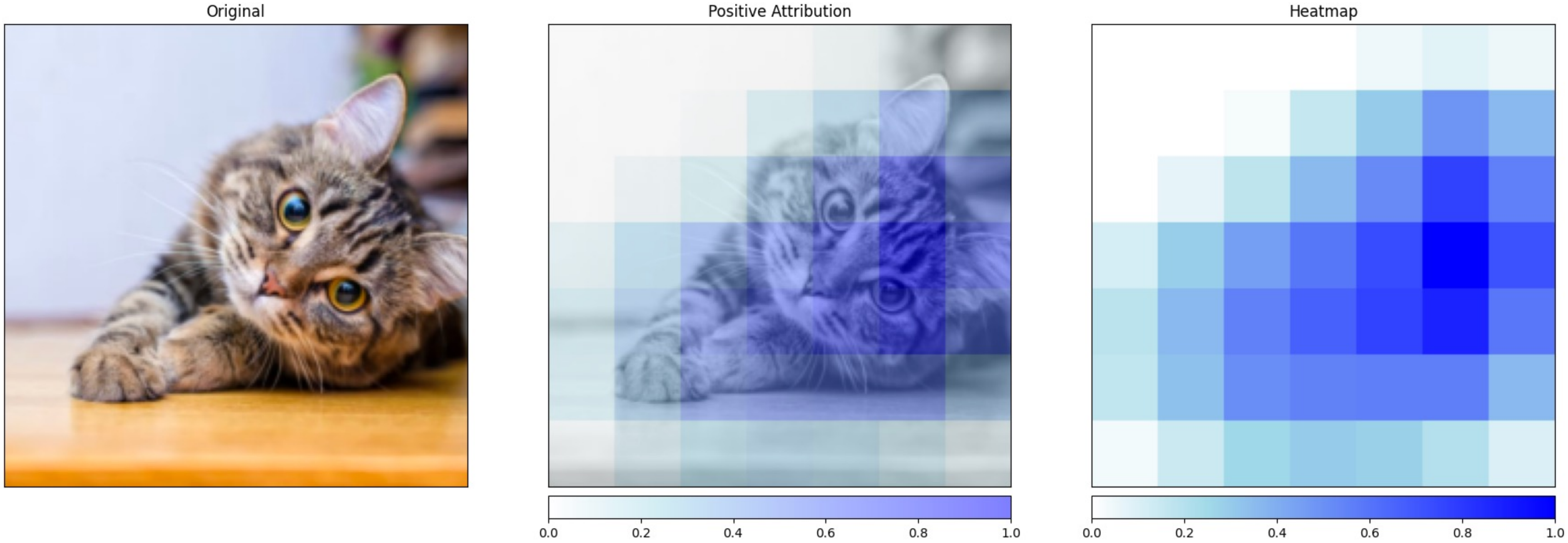
c. Multiply the average gradient for each channel by the layer activations and apply a ReLU

$$Layer\ –R^c = ReLU(\sum_k \alpha_k A^k)$$

ReLU since only interested in the features that have a positive influence on the class, i.e. pixels whose intensity should be increased to increase class c

GradCAM - Relevance - Example

- $Layer - R^c$ in a **coarse heatmap** of the same size as the convolutional feature maps
- Often we upsample it and view as mask to the input



Guided GradCAM

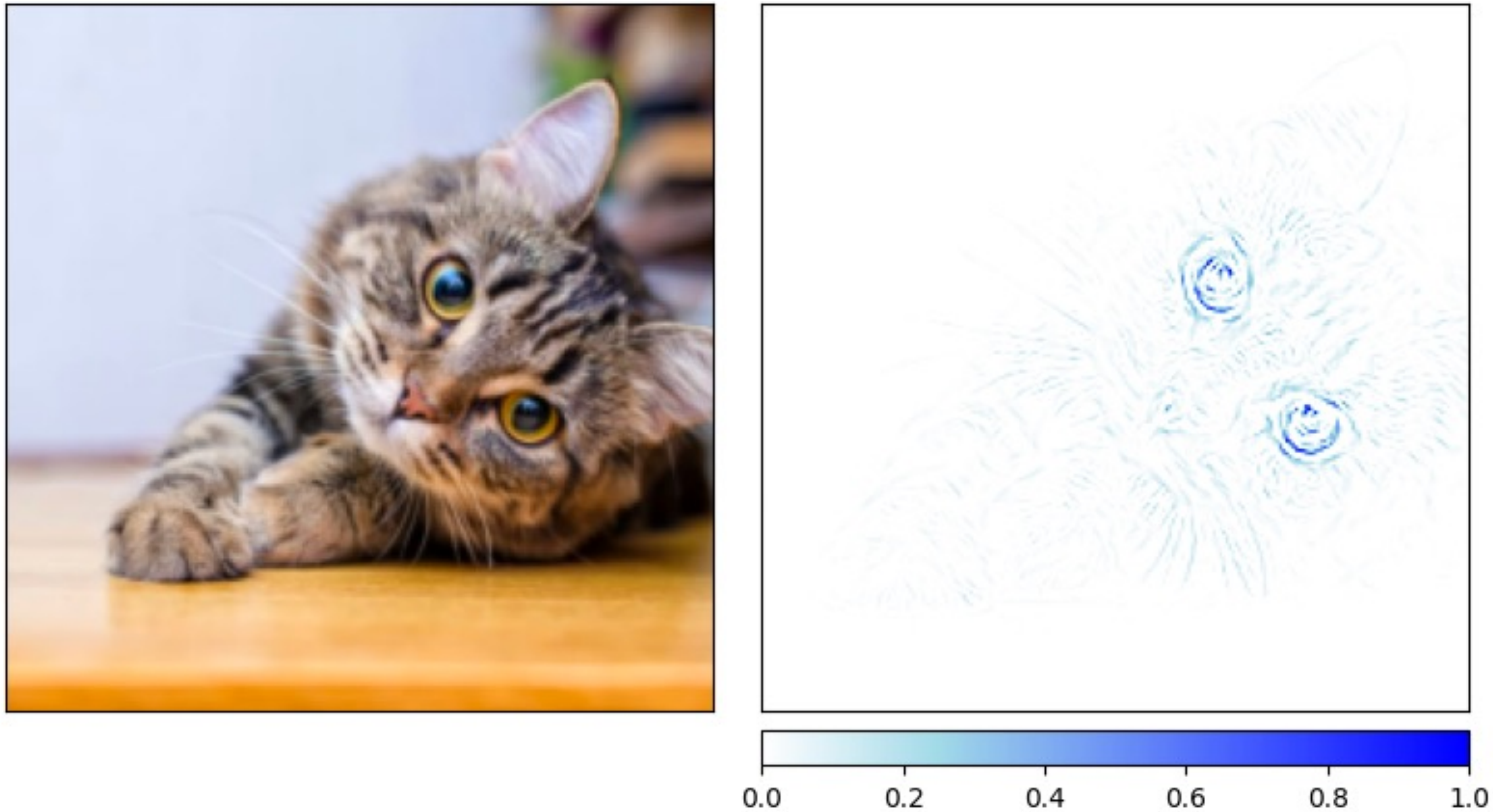
- Grad-CAM produces coarse importance as the last convolutional feature maps have a coarser resolution compared to the input image
- We may want to have a per-pixel importance

- **Idea**

Combine Grad-CAM explanation and the explanation from another attribution method, such as Vanilla Gradient, by multiplying element-wise

$$\text{Guided Grad-CAM} = upsample(Layer - R^{c}_{GRADCAM}) \odot R^{c}_{other\ method}$$

Guided GradCAM - Example



Integrated Gradients

Propose two **axioms**: sensitivity and implementation variance

Sensitivity.

If two inputs x and x’ differ only in one feature A_i but have different predictions, then the feature A_i should be given a non-zero attribution

Example

x = [1, 0, 1] → f(x) = class 0

x’ = [1, 1, 1] → f(x) = class 1

Integrated Gradients

Axioms.

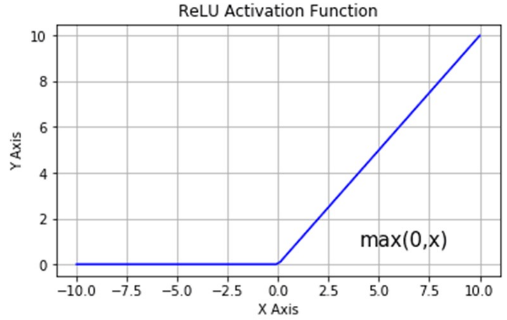
Implemetation invariance

If two models f anf f’ have identical input/output behvior, then the attributions for M and M’ should be identical.

Gradients X Input fails sensitivity

- f(x) = 1-ReLU (x) = 1-max(0, 1-x)

- Example
  - f(0) = 1-max(0, 1-0) = 0
  - f(2) = 1-max(0, 1-2) = 1



$$\nabla RELU'(x) = \begin{cases} 1\ if\ x > 0 \\ 0\ if\ x \leq 0 \end{cases}$$

Since we have different outputs, for the sensitivity axiom, we should have **different attributions**

$$\textbf{InputXGradient}(f, x) = \nabla f(x) \cdot x$$

$$\textbf{InputXGradient}(f, 0) = 1 \cdot 0 = 0$$

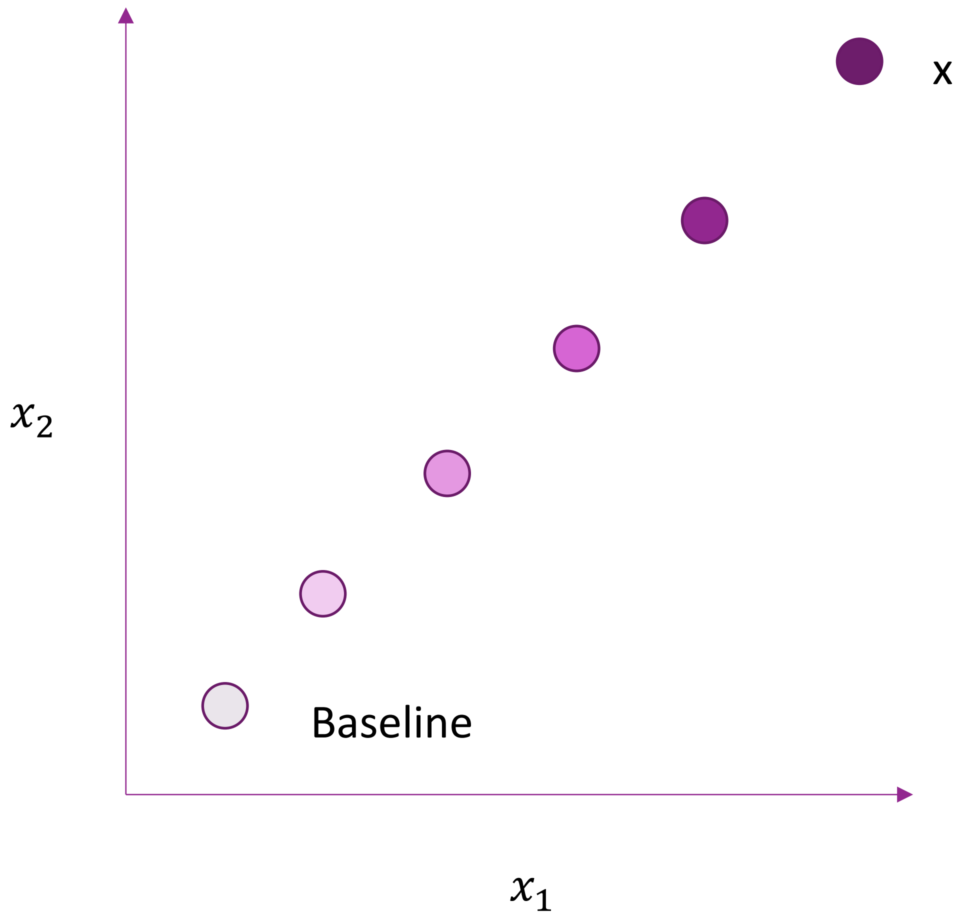
= → Failed test!

$$\textbf{InputXGradient}(f, 2) = 0 \cdot 2 = 0$$

$$\nabla f(x) = \begin{cases} 1\ if\ x < 1 \\ 0\ if\ x > 1 \end{cases} = \max(0, sign(1 - x))$$

$$sign(x) = \begin{cases} -1\ if\ x < 0 \\ 0\ if\ x = 0 \\ 1\ if\ x > 0 \end{cases}$$

Integrated Gradients



- Compare x with a **baseline**
  - No information, e.g., zero vector
- Interporlate between the point of this baseline and the input x
- Take the gradient with respect to each interpolated input
- Compute the average of these gradients
  - Give us the feature importance

Animation from 'Feature Attribution' | Stanford CS224U Natural Language Understanding | Spring 2021

Integrated Gradients

- Let x be the instance to explain and x'a baseline input.
- Let α be interpolation constant to perturb features by

$$IntegratedGrads_i(x) = (x_i - x_i') \times \int_{\alpha=0}^{1} \frac{\partial f\big(\, x' + \alpha \times (x - x')\big)}{\partial x_i}\, da$$

We actually compute the numerical approximation instead of the integral

Let k be a scaled feature perturbation constant and m the number of steps

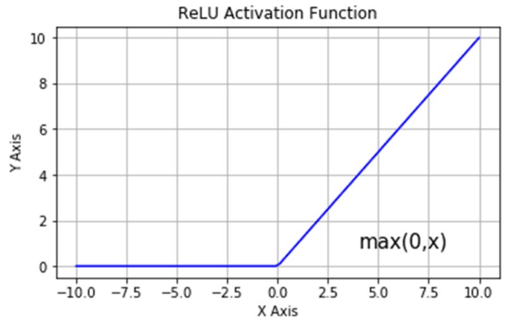
$$IntegratedGrads_i^{approx}(x) = (x_i - x_i') \times \frac{1}{m} \times \sum_{k=1}^{m} \frac{\partial f(\, x' + \frac{k}{m} \times (x - x'))}{\partial x_i}$$

Integrated Gradients - steps

$$IntegratedGrads_i^{approx}(x) = \overset{5}{\overline{(x_i - x_i')}} \times \overset{4}{\overline{\frac{1}{m} \times \sum_{k=1}^{m}}} \overset{3}{\overline{\frac{\partial f(\overset{2}{\overline{x' + \overset{1}{\overline{\frac{k}{m}}} \times (x - x')}})}{\partial x_i}}}$$

1. Consider multiple perturbations
2. Interpolate inputs between baseline x’ and the input x
3. Compute the gradients for each interpolated input
4. Compute the average – approximation of the integral
5. Scale to remain in the original space

Integrated gradients and sensitivity



$$\nabla RELU'(x) = \begin{cases} 1 \; if \; x > 0 \\ 0 \; if \; x \leq 0 \end{cases}$$

$$\nabla f(x) = \begin{cases} 1 \; if \; x < 1 \\ 0 \; if \; x > 1 \end{cases} = \max(0, sign(1 - x))$$

- **Example**
  - f(x) = 1-ReLU (x) = 1-max(0, 1-x)
  - f(0) = 1-max(0, 1-0) = 0
  - f(2) = 1-max(0, 1-2) = 1

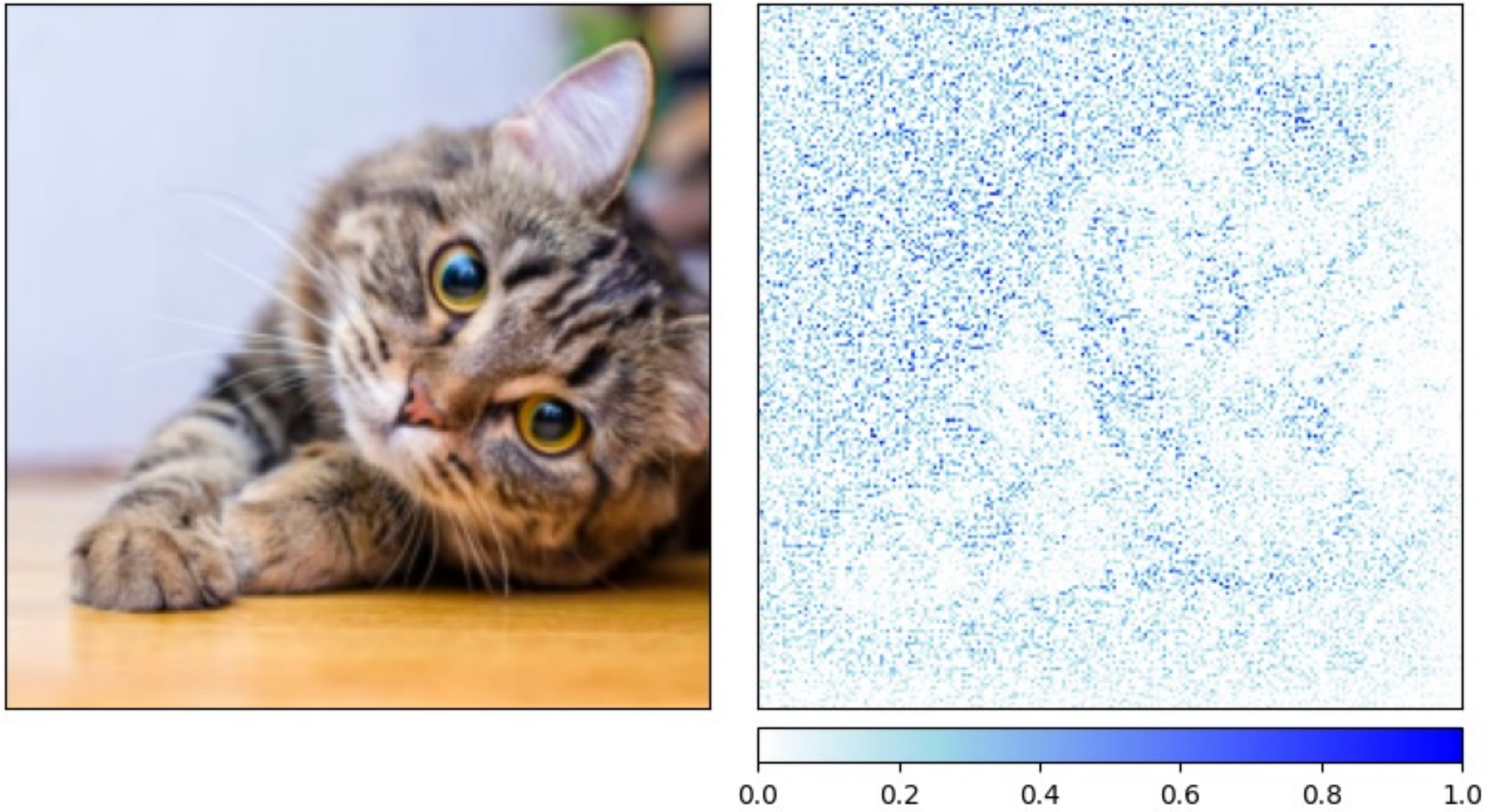
Since we have different outputs, for the sensitivity axiom, we should have **different attributions**

IntegratedGradient(f, x, x') = $(x_i - x_i') \times \sum_{k=1}^{m} \frac{\partial f(\, x' + \frac{k}{m} \times (x - x'))}{\partial x_i} \times \frac{1}{m}$

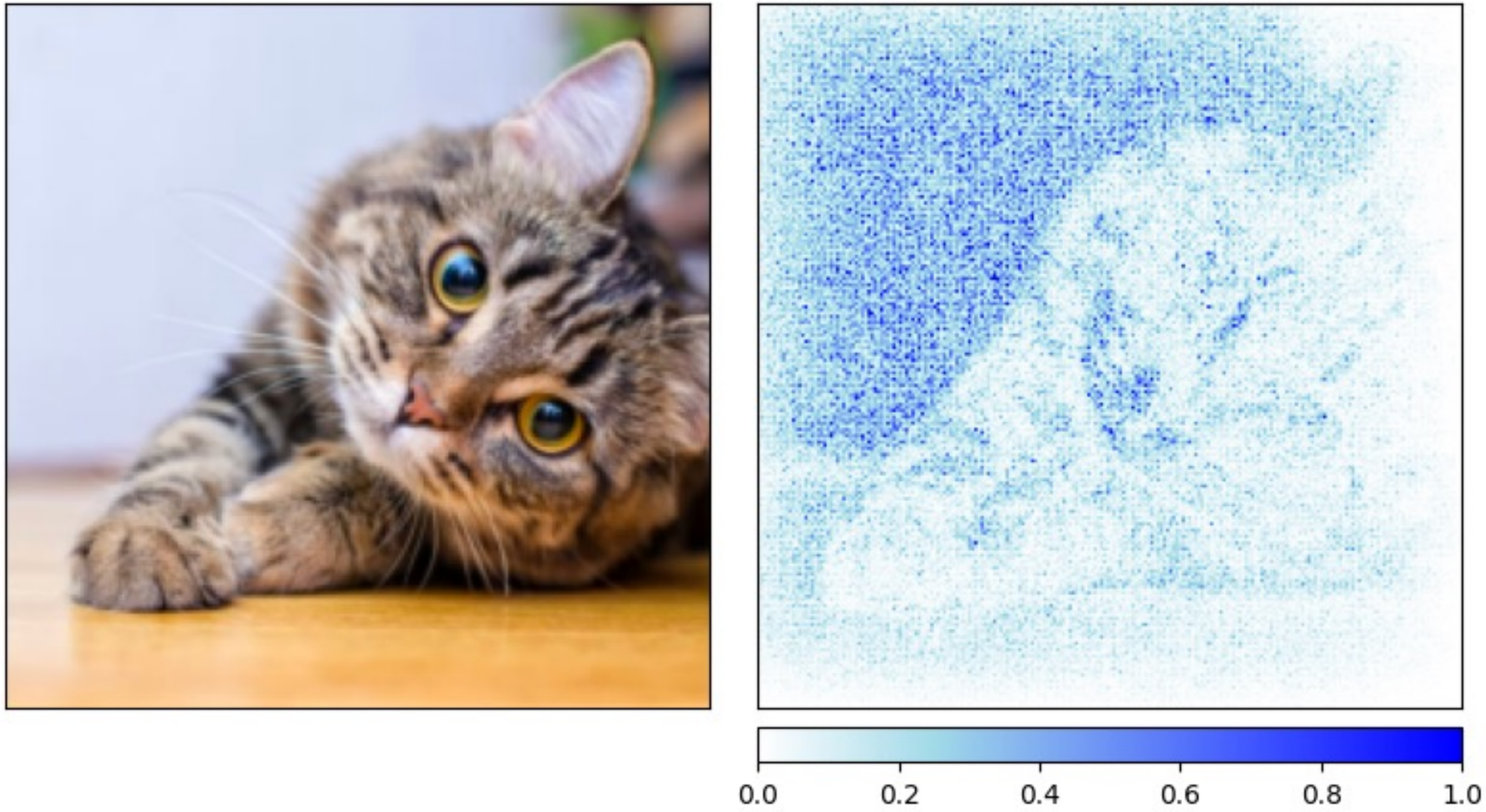
IntegratedGradient$(f, 2, 0) \; = (2 - 0) \frac{1}{m} \sum(\max(0, sign(1 - 0.00) + \max(0, sign(1 - 0.02)) + .. + \max\big(0, sign(1 - 2)\big) \approx 1$

IntegratedGradient $(f, 0{,}0) \approx 0$

Integrated Gradients - Example



Integrated Gradients + SmoothGrad - Example



It is more computational expensive! Locally only on 5 samples..

Advantages

- Efficiency
  - Many gradient-based methods are computationally efficient

- Multiple approaches

- Effective visualization via saliency maps

Limitations

- Some methods do not satisfy the sensitivity axiom
  - Methods insensitive to model and data.
  - Explainers or edge detectors simply highlight color changes in images?

- Sensitivity to Perturbations
  - Methods may be sensitive to small changes in input data, leading to unstable explanations

- Gradient Vanishing/Saturating

- Different approach, different explanations..
  - Which one to trust?
  - Need for evaluation approaches..!