Warsaw University of Technology

FACULTY OF
MATHEMATICS AND INFORMATION SCIENCE

# Bachelor's diploma thesis

in the field of study Data Science

Automated explanation selection for convolutional neural networks

Agata Kaczmarek

student record book number 305751

Mateusz Stączek

student record book number 305757

Paweł Wojciechowski

student record book number 298920

thesis supervisor

mgr inż. Hubert Baniecki

consultation

dr hab. inż. Przemysław Biecek, prof. uczelni

WARSAW 2023

**Abstract**

Automated explanation selection for convolutional neural networks

More and more complex neural networks are being used in computer vision and it creates a need to explain their decision processes. Therefore, many explanation methods designed for convolutional neural networks have been created, each with specific limitations and ways of explaining model's operation. Researchers and users of machine learning models may face difficulties when willing to explain their models because each explanation method is somewhat unique and the selection of a single one is not an easy task. Also, comparing different explanation method results is a time-consuming work. However, various metrics for evaluating explanation methods were developed that could help to answer the hard question "Is this a good explanation method for this model?"

Due to these circumstances, we develop the AutoeXplain Python package that automates the selection of an explanation method for convolutional neural network models. Users of this package do not need to have advanced knowledge about explanation methods or evaluation metrics and can easily obtain the explanation method that is most suitable for their case according to available metrics. The results of the evaluation can be placed in a generated report to get convenient insight into the evaluation process. Therefore, the AutoeXplain package saves time and lowers the entry threshold for comparing explanation methods. The results of the analysis can be also a baseline for further research of explanation approaches or be a part of the model's documentation that shows its interpretability aspects.

**Keywords:** explainable artificial intelligence, Python, computer vision, image classification, explanation evaluation

**Streszczenie**

Automatyczny wybór metody wyjaśnień dla konwolucyjnych sieci neuronowych

Sieci neuronowe przeznaczone do zadań wizji komputerowej stają się coraz bardziej złożone, co powoduję potrzebę wyjaśnienia ich procesu decyzyjnego. Powstało już wiele metod wyjaśnień przeznaczonych dla konwolucyjnych sieci neuronowych, a każda z nich ma własne ograniczenia i stara się wyjaśnić działanie modelu w różny sposób sposób. Badacze oraz użytkownicy modeli uczenia maszynowego mogą napotkać trudności, gdy chcą wyjaśnić swój model, ponieważ każda metoda wyjaśnień jest na swój sposób inna, a wybranie jednej z nich nie jest łatwym zadaniem. Ponadto, porównywanie wyników różnych metod wyjaśnień może być czasochłonną czynnością. Jednakże, powstało wiele różnych metryk służących do ewaluowania działania metod wyjaśniających modele. Metryki te pomagają w poszukiwaniu odpowiedzi na pytanie "Czy to jest dobra metoda wyjaśnień dla danego modelu?"

Wobec tego, w języku Python zaimplementowaliśmy pakiet AutoeXplain, który automatyzuje wybór metody wyjaśnień dla konwolucyjnych sieci neuronowych. Użytkownicy tego rozwiązania nie muszą posiadać zaawansowanej wiedzy na temat metod wyjaśnień modeli uczenia maszynowego i mogą uzyskać rozwiązanie dla podanego problemu, której jest najlepsze względem dostępnych metryk. Wyniki ewaluacji mogą zostać umieszczone w wygenerowanym raporcie by w przystępny sposób mieć wgląd w szczegóły procesu ewaluacji. Dzięki temu, pakiet AutoeXplain pozwala zaoszczędzić czas podczas badań i umożliwia porównanie metod wyjaśnień szerszemu gronu użytkowników. Wyniki analizy mogą być punktem startowym dla dalszych poszukiwań metody wyjaśnień, bądź być częścią dokumentacji modelu dotyczącej jego wyjaśnialności.
**Słowa kluczowe:** wyjaśnialna sztuczna inteligencja, Python, wizja komputerowa, klasyfikacja obrazu, ewaluacja metod wyjaśnień

# Contents

# 1. Introduction

Machine learning is widely used in various domains including medicine and sociology [1], [2]. Models with the highest performance scores, especially deep learning neural networks happen to have such a complex structure, that they are difficult to understand by humans [3]. In some domains, it is crucial to know, if the model learned accurately – for example in medicine, if it can find cancer cells correctly recognizing key features in data – at the end, human life depends on the model decision [2]. In that case, it is also important for doctors to know, how much they can trust the results given by the model [4]. To solve this problem, methods in the field of explainable artificial intelligence (XAI) are introduced [5]–[7]. Although a wide range of explanation methods to choose from may seem beneficial, it could be a challenge to decide, which XAI method should be selected to be used in a given scenario.

## 1.1. Motivation

There are two types of evaluations of XAI methods: objective and subjective. An objective evaluation can be to check if the result of an explanation method fulfills certain axioms [8]. On the other hand, an example of a subjective type is to ask humans to choose which result of an XAI solution, called attribution, seems to be the best for them. In [7] authors tested the results of the Grad-CAM method in the survey asking humans for their opinion. 43 workers of Amazon Mechanical Turk answered detailed questions about objects on images and their way of choosing the most important object on the image. Some of the advantages of that kind of survey are that they can check which attribution is more aesthetically pleasing [7]. However, there are also important disadvantages of these methods – this method is time-consuming, but more importantly, the selection of participants in that kind of survey can determine the results. This is because of the experience, beliefs, and bias each of the users have [9], [10].

Some of before mentioned problems can be solved, when using automatic methods instead of asking humans. For example, automatic selection is faster than asking humans. Additionally, the evaluations that are of objective type, have many advantages including stable reference point, which does not change compared to human opinion thus allowing to objectively compare and improve explanation methods. Secondly, that type of evaluation gives a stable theoretical basis to prove the value of a chosen explanation method [8]. Additionally, automatic selection of the best method simplifies the work of the data scientist, helping out in for example checking which XAI solution is compatible with which machine learning model [11].

The main goal of this thesis is to develop a Python package that performs an automatic selection of the explanation method for convolutional neural networks (CNN) used in image classification. Selection is based on the objective evaluation made with beforehand implemented metrics [8], [12], [13], [14], which leverage all mentioned benefits.

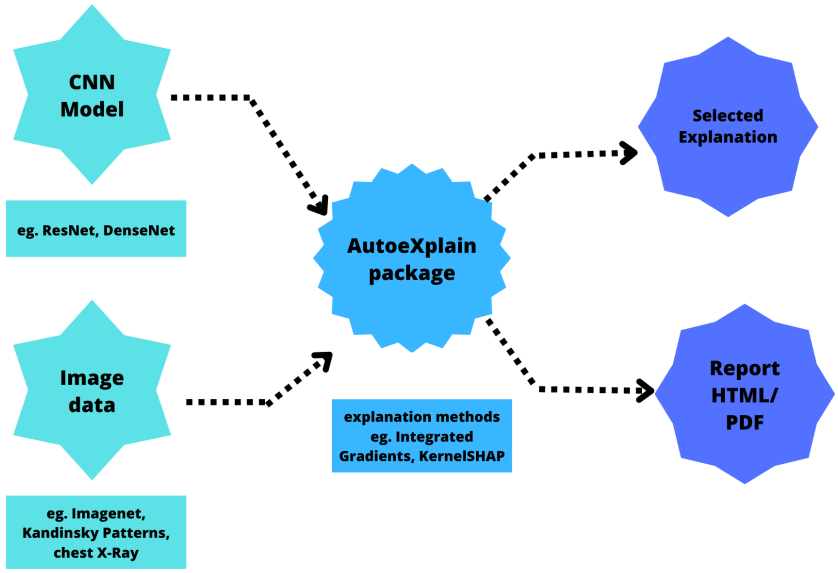



Figure 1.1: The AutoeXplain package. Its goal is to automatically select the best explanation method for a given problem that contains a CNN model and image dataset. The AutoeXplain package benchmarks four different explanation methods with the use of four evaluation metrics, focusing on different aspects of the explanations: faithfulness, robustness, and complexity (more details in Section 2.2).

## 1.2. Contribution

To solve the problem of selecting explanation methods for convolutional neural networks for image data, we create a Python package named AutoeXplain. The contribution of this thesis is as follows:

1. The package automatically selects the best explanation method for a given problem, defined by provided image dataset and CNN model, and according to the available evaluation metrics. It selects one of four different evaluation methods available in the AutoeXplain package. It also accelerates the feedback loop, as it is enough to use four lines of code to run the benchmark.

2. Afterwards, the package creates both HTML and PDF reports, which include details about the evaluation process of XAI solutions and provided model itself. This feature is crucial because it provides more detailed information about the results of consecutive methods and metrics, which can help in further work with classification problems.

According to our knowledge, currently, there is no such possibility in Python to automatically select explanation methods for CNNs for image data. The work of this package is shown in three use cases, each including a different type of image data. The first one contains synthetic images, generated based on Kandinsky Patterns work [15], the second dataset focuses on Chest X-ray scans [16], the third one is Imagenette – a subset of the ImageNet dataset [17]. For each of those three datasets, a machine learning model has been trained and used as part of the use case for the AutoeXplain package.

The content of this work is as follows: in Section 1.3 we compare work related to this thesis, then in Section 2 we describe the most important concepts: definitions of explanation methods and evaluation metrics used in the package. Following, Section 3 introduces the package and its implementation with an example of usage. Later we describe experiments conducted on the created package in Section 4. Section 5 concludes the thesis with remarks on future work.

## 1.3. Related work

This section describes related work on topics connected to our work: explanation methods for deep neural networks, evaluation of attribute-based explanation methods, and other proposed approaches for selecting the best explanation techniques.

### 1.3.1. Explanation methods for deep neural networks

There are various local methods for explaining deep neural networks designed for image data. A review of deep learning explanation methods [18] specifies three explanation families: interpretable local surrogates, occlusion analysis, and gradient-based techniques.

Interpretable local surrogates use self-explanatory models (like logistic regression) as decision functions locally in the neighbourhood of explained example. This approach is used in techniques like Local Interpretable Model-agnostic Explanations (LIME) [19]. This model-agnostic local method is designed to estimate feature importance through coefficients of a local linear regression model. This linear model is trained on data $\{(x_i.y_i)\}$, where $x_i$ being input date, and $y_i$ being target. $x_i$ are perturbed versions of examined data instance, by zeroing a subset of features. $y_i$ is the prediction of the examined model done on $x_i$. Such an approach can be used for various modalities of data. For image data, pictures are divided into contiguous segments (superpixels) and these segments are treated as single features. Another example of an interpretable surrogate is Anchor [20] - instead of linear models like in LIME, surrogate interpretable models are IF-THEN decision rules called “anchors”. Decision rules are determined by using reinforcement learning techniques together with a graph search algorithm which helps to reduce computation time. Image perturbations have a different form than in LIME. Instead of removing some segments, a subset of segments is pasted from other images. Interpretable local surrogates methods do not rely on gradients of the examined model, so they avoid gradient difficulties in explaining models [18], e.g. the “shattered gradient” effect [21], in particular in very deep networks.

Perturbation analysis measures the impact of the modifying input image on the model's output. Perturbations are usually performed on patches (superpixels) of images [5], [22], [23]. The outcome of such as procedure may be a heatmap called an attribution map or saliency map. They represent which regions of images are significant for the prediction made by the model. One of these explanations is Occlusion Sensitivity [23]. It works by removing square patches of the image and measuring changes in model prediction. Removal is done by replacing pixels with baseline color, e.g. black (zero values), grey (middle values of pixels), or the mean color of the image. Another example is RISE [22] algorithm. Images are perturbed by element-wise multiplication with randomly generated masks with values $[0, 1]$. Each perturbed is passed through the model and resulting probabilities are saved. The final saliency map is a linear combination of masks and obtained probabilities. Previously mentioned LIME [5] is also a perturbation-based method.

Gradient-based techniques analyze gradient flow up to input space. Usually, these approaches result in heatmaps indicating the importance of pixels for a given image prediction. Integrated Gradients [6], Saliency [24], Grad-CAM [7] are examples of gradient-based methods. Their functionalities are described in detail in Section 2.1. The created tool utilises a diverse set of explanation methods, which come from all mentioned families.

### 1.3.2. Evaluation of feature attribution methods

Recently, the evaluation of XAI methods is a frequently raised issue [9], [25]–[29]. There is a proposed taxonomy [30] for the evaluation of XAI solutions in general. It specifies three kinds of approaches for the evaluation of XAI algorithms, each one is more costly but more specific for a problem than the previous one: functionality-grounded evaluation, human-grounded evaluation, and application-grounded evaluation. Human-grounded evaluations are based on the user studies conducted on users who are not related to the domain of the end application. This form inherently brings subjectivity in some way. Application-grounded evaluations are similar to human-grounded evaluations, but they are conducted with experts from the domain of the application problem, often end-users of the final solution.

Functionality-grounded evaluations do not require human involvement in evaluation, they base on formally defined "proxies" [12] of desiderata for an ideal explanation method. Due to the low cost and time required, it allows us to evaluate and compare many XAI methods. This kind of evaluation is also free from the subjectivity of the user, as evaluation comes from objective quantitative metrics [28].

These qualities make functionality-grounded evaluations well-suited for our issue – creating an automated tool that needs minimal human input. Additionally, in this area of evaluation, many quantitative metrics were proposed, which measure different aspects of the explanation method. They can quantify the stability of methods [8], [12], their fidelity to the behavior of examined model [12], [13], [31], [32] or robustness against adversarial data [29]. They are especially suitable for feature attribution methods that are used in our work.

Our solution is closely bound to the Quantus Python [33] package. It implements over 25 evaluation measures for feature attribution explanations. Our work is built upon this software and extends its functionality with automatic explanation selection for CNN. Specifically, we

develop a tool with reporting functionality that makes explanations and their evaluation more accessible for machine learning users.

### 1.3.3. Other approaches for selection attribution-based methods

To the best of our knowledge, while there are multiple ways of evaluating XAI algorithms, there are not so many strategies or frameworks for XAI method selection, especially those designed for deep learning models and image data. There is a proposed generic methodology [34] for selecting and implementing XAI methods. The methodology uses explanation methods defined ad–hoc and matches them against user requirements during the analysis step. The properties of XAI methods and the questionnaire that helps to reveal end-user needs are enclosed in that paper.

During work on this thesis AutoXAI [11] was presented. It is a framework for selecting explanation methods. As explanation methods have various parameters to be set prior to their use, AutoXAI has functionality for searching optimal (with respect to evaluation metrics) setup of parameters for a given task. It assumes that the user provides the context for evaluation, such as model, data, and optionally method's hyperparameters' bounds. Afterward, explanation methods are evaluated against several evaluation metrics. Parameters of explanation methods can be optimized with respect to calculated metric values. However, provided code is only for experiments presented in the article, which are focused on tabular data and classic machine learning models. Moreover, it is limited to two explanation methods – KernelSHAP and LIME. Our solution is designed for CNNs that are used with image data. For now, it uses 4 explanation techniques, however, the modularity of AutoeXplain's API allows for easy extension with other local attribution-based methods.

# 2. Theory

In this thesis, we aim to select explanation methods for a special type of neural networks – convolutional neural networks. In general, neural networks contain a high number of computational nodes, called neurons, which are connected with each other in specific sequences. The neural networks can be applied to classification problems. The parameters of such a network are optimized with the use of back-propagation – gradient-based procedure [35]. To explain better, how a neural network works, Figure 2.1 presents the schema of a sample one. This neural network has four neurons in the input layer, two hidden layers with two neurons each and one neuron in an output layer.

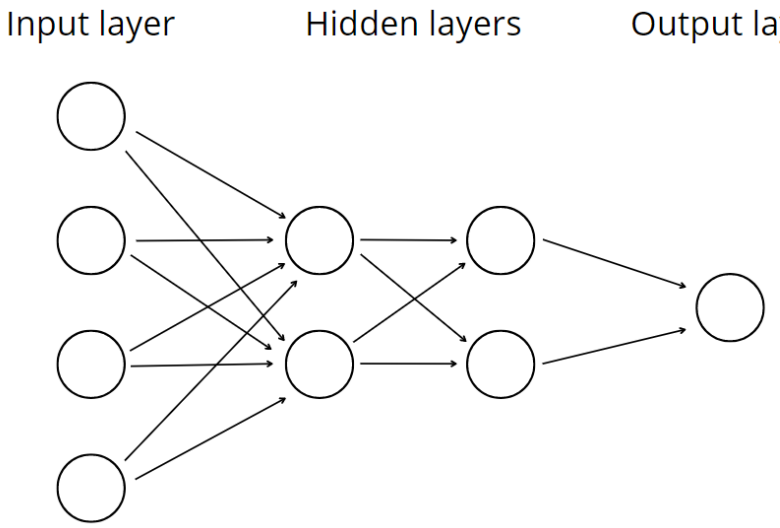



Figure 2.1: Schema of a simple neural network with the input layer, two hidden layers and an output layer. Schema created by the authors.

During this work, we analyze one of the types of neural networks – convolutional neural networks. CNNs are designed to process images. They contain convolutional, pooling and fully-connected layers architecture. A convolutional layer is responsible for calculating the scalar product of input and weights for neurons connected in local regions. A pooling layer reduces the number of activations on which all other calculations are performed. Fully-connected layer behaves as a simpler neural network, calculating scores for classes, to be used for final predictions. The schema of a simple CNN for a classification problem is shown in Figure 2.2.

Over the last decades, various CNN architectures were introduced to solve image classification tasks. They vary in activation functions, number of layers, number of neurons per layer and other parameters. ResNet [37] and DenseNet [38] are examples of CNNs, which have been widely used

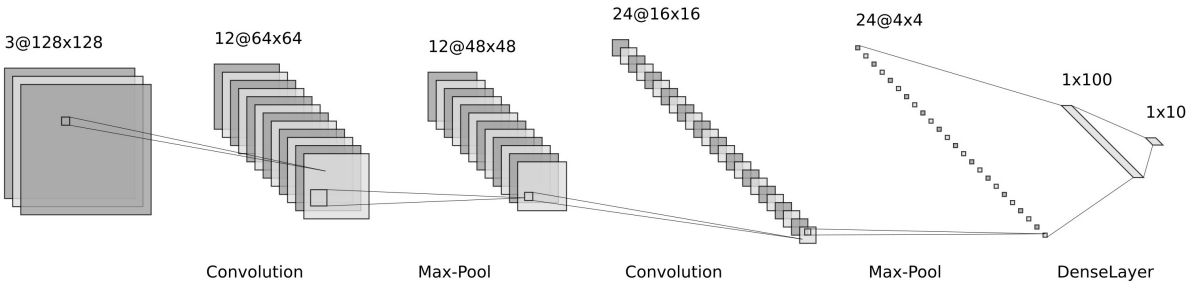



Figure 2.2: Schema of an exemplary CNN architecture. Image generated by the authors using NN-SVG tool [36].

in various classification tasks. DenseNet supersedes ResNet, as it implements dense concatenation of layers and is more efficient [39].

## 2.1. Explanation methods for convolutional neural networks

An explanation method is an algorithm that tries to explain the decision process of a machine learning model. In our work, we focus on local feature attributions – highlighting various parts of a certain image, which influence on the model's prediction we want to understand [40]. Each time an attribution is computed for a single image. Furthermore, here, the end-user of the explanation method is a human, so the generated attributions cannot be too complex. Therefore, attribution can be visualized with a heatmap highlighting the local feature attributions. Figure 2.3 contains attributions computed for a sample input image. They were computed using 4 explanation methods: Grad-CAM [7], Integrated Gradients [6], KernelSHAP [5] and Saliency [24].

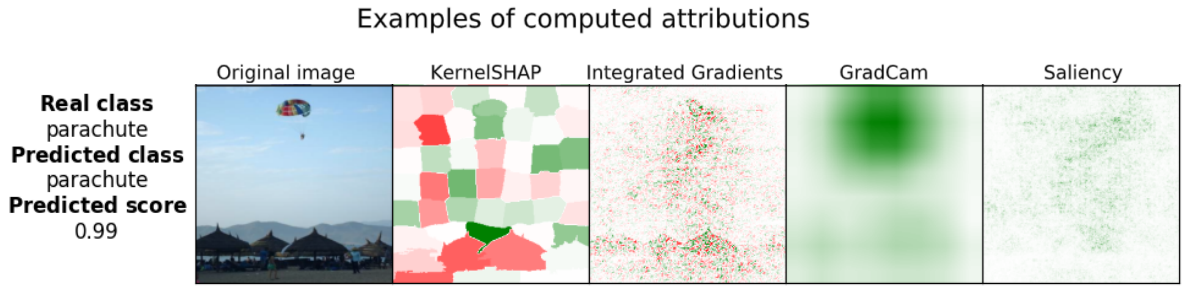



Figure 2.3: Sample attributions generated for the original input image using four explanation methods.

In heatmap visualizations in our work, negative values are visualized with red pixels, positive with green, and values close to 0 are white. In short, white pixels did not have much influence on the model's prediction, green pixels contributed to model's prediction for the given class and red pixels had the opposite impact. It is important to note that some explanation methods,

such as Saliency, do not distinguish between positive and negative contributions. Instead, such explanation methods measure only the magnitude of an attribution. On the other hand, Grad-CAM's [7] paper suggests using ReLU activation function to remove negative contributions.

### 2.1.1. Grad-CAM

Gradient–weighted Class Activation Mapping (Grad-CAM) [7] is an explanation method for CNNs that is based on activation values from some selected layer that are weighted by gradients for the selected class. First, the selected image is forward propagated and CNN outputs are acquired. Next, after calculating the gradients for these outputs, all of these gradients are changed to 0 except of the gradient for the selected class. Finally, such gradients are backpropagated until they reach the selected layer where they are said to be the importance of each channel at that layer. The attribution is a rectified linear combination of activations from the selected layer where the importance plays the role of weights.

Let $r$ be the selected layer and all of the channels will come from this layer. Also, let $x_m \in \mathbb{R}^n$ be the m–th image from the dataset for which the attribution is computed. Then, let $\gamma^c_{t,m}$ be this importance for some channel $t$ from the layer $r$ that was computed for the class $c$ and the m–th image. Let $A^t_m$ be activation values for the channel $t$. The attribution $v^c_m$ is obtained using:

$$v^c_m = \text{ReLU}\left(\sum_t \gamma^c_{t,m} A^t_m\right). \tag{2.1}$$

ReLU stands for a rectified linear unit activation function represented as $f(x) = \max(x, 0)$ for every element of the input. ReLU suppresses the negative gradients which may originate from other classes. Therefore, the attribution heatmap should be more clear and it should not highlight elements of other classes. It is worth noting, that the attribution $v^c_m$ has a much lower resolution than the original image but can be interpolated back to the same size.

The paper that introduced Grad-CAM suggests that the selected layer $r$, for which the Grad-CAM produces the heatmap, should be the last convolutional layer of the CNN. Also, this recommendation is based on other works mentioned by this paper. Furthermore, in an example in which Grad-CAM was used on different layers, the best results were obtained when the last layer is selected.

### 2.1.2. Saliency

The Saliency [24] explanation method is based on computing gradients – the basic idea is to approximate CNN's output for a given class in the neighbourhood of the image using a linear approximation and interpret the coefficients vector as an importance vector for all pixels.

Let $F : \mathbb{R}^n \to \mathbb{R}^k$ be a CNN model that classifies input images into $k$ classes. Next, let $x_m \in \mathbb{R}^n$ be the m–th input image. Also, let $T^c$ denote logit for the given class $c$, before the softmax layer. Then, the linear approximation of $T^c(x)$ in the neighbourhood of $x_m$ can be made using first-order Taylor expansion:

$$T^c(x) \approx v^T \times x + b \tag{2.2}$$

for some $b \in \mathbb{R}$. Here, the $v_m^c \in \mathbb{R}^n$ represents the importance of each pixel in the image $x_m$ and is calculated as

$$v_m^c = \frac{\partial T^c}{\partial x}\Big|_{x_m}. \tag{2.3}$$

For images with more than a single channel, for example, RGB image, the value of $v_m^c$ is taken for each pixel as a maximum across all 3 channels.

The main advantage of this method is its speed as it requires only a single backpropagation pass for each image to calculate the gradients. Also, it does not require any modifications to the network's architecture or structure. The method is class-discriminative too.

An implementation of this explanation method may return an absolute value of the vector $v_m^c$. In such case, it does not distinguish between positive and negative attributions but it is not an issue – this is the very nature of the $v_m^c$. By design, it is only to show the magnitude of change in model's prediction after changing the given pixel's value.

### 2.1.3. Integrated Gradients

An example of an explanation method that is based on approximating gradients, is Integrated Gradients [6]. It analyzes multiple slightly modified versions starting with the baseline image and aggregates the results. The baseline image should be such that the model's prediction for such an image is close to zero. For CNNs, the baseline image can be set to a solid black color.

Let $F : \mathbb{R}^n \to \mathbb{R}^k$ be a neural network in a classification task for $k$ classes, $x_m \in \mathbb{R}^n$ be the m–th input image and $x'_m \in \mathbb{R}^n$ be the baseline image. Also, let $F^c$ be model's prediction for the given class $c$. For the baseline image $x'_m$, we always use a zero vector which corresponds to a black image. Each pixel $v_{m,i}^c \in \mathbb{R}$ of the attribution $v_m^c \in \mathbb{R}^n$ represents an approximation of

$$v_{m,i}^c \approx (x_{m,i} - x'_{m,i}) \cdot \int_{\alpha=0}^{1} \frac{\partial F^c(x'_m + \alpha(x_m - x'_m))}{\partial x_{m,i}} \mathrm{d}\alpha \tag{2.4}$$

where $\frac{\partial F^c(x_m)}{\partial x_{m,i}}$ is a gradient of the neural function at i–th dimension (pixel) of m–th image. This integral is approximated using the following sum with a recommended number of steps $D$ being between $D = 20$ and $D = 300$:

$$v_{m,i}^c = (x_{m,i} - x'_{m,i}) \cdot \frac{1}{D} \sum_{d=1}^{D} \frac{\partial F^c(x'_m + \frac{d}{D}(x_m - x'_m))}{\partial x_{m,i}}. \tag{2.5}$$

The images (represented as points in $\mathbb{R}^n$) in between the original image $x_m$ and the baseline image $x'_m$ lie on the shortest path between these 2 points. This property makes this explanation method a *path method* and it was proved that such methods satisfy desirable axioms such as implementation invariance, sensitivity, linearity, and completeness, further described in the original article [6].

### 2.1.4. KernelSHAP

Explanation method KernelSHAP [5] is part of the SHapley Additive exPlanation (SHAP) framework that aims to measure feature importance by approximating Shapley values. Ker-

nelSHAP for image classification tasks is based on LIME [19] where the parameters are selected in a more specific way, as described below.

LIME explanation method locally approximates the k–class classification model $F : \mathbb{R}^n \to \mathbb{R}^k$ in the neighborhood of the model's prediction for m–th input $x_m \in \mathbb{R}^n$ using a linear equation:

$$g(x'_m) = v_{m,0} + \sum_{l=1}^{L} v_{m,l} q^l_m. \tag{2.6}$$

Here, $g$ is a linear explanation model with $v_m$ as coefficients and $q_m \in \{0,1\}^L$ as an input that is a simplified version of the original input $x_m \in \mathbb{R}^n$. There are $L$ features denoted by $q^1_m$ to $q^L_m$. Each feature $q^l_m$ can either be 0 or 1 and means whether a given l–th feature is present in the simplified version of the input $x_m$ or not. Each coefficient $v_{m,l}$ also denotes the importance of the l–th feature.

In the case of LIME and CNNs, a segmentation algorithm is used to divide an image into a set of $L$ superpixels. Feature $q^l_m$ corresponds to a superpixel which is a group of pixels. When $q^l_m$ is set to 0, pixels belonging to the corresponding superpixel in the original input image $x_m$ are changed to the color black. This is supposed to mean a lack of this feature when a CNN analyses such a modified image.

LIME as an algorithm has a few important parameters which are chosen heuristically. These parameters are: $\mathcal{L}$ – squared loss function, $\pi_{q_m}$ – weighting kernel, $\Omega$ – regularization term. Other elements of the input are $F$ – explained model and $g$ – explanation method linear model. The algorithm is based on minimizing the following formulae over possible explanation models $g \in G$:

$$\underset{g \in \mathcal{G}}{\operatorname{argmin}}\ \mathcal{L}(F, g, \pi_{q_m}) + \Omega(g). \tag{2.7}$$

KernelSHAP defines the loss function $\mathcal{L}$, the weighting kernel $\pi_{q_m}$ and the regularization term $\Omega$ as follows:

$$\mathcal{L}(F, g, \pi_{q_m}) = \sum_{q'_m \in Q_m} \left[F(h^{-1}_{x_m}(q'_m)) - g(q'_m)\right]^2 \pi_{q_m}(q'_m), \tag{2.8}$$

$$\pi_{q_m}(q'_m) = \frac{L-1}{\binom{L}{|q'_m|})|q'_m|(L-|q'_m|)}, \tag{2.9}$$

$$\Omega(g) = 0. \tag{2.10}$$

Here, $q'_m \in Q_m$ represents a vector from the set $Q_m$ of all vectors $q'_m$ such that their non–zero elements are a subset of non–zero elements of $q_m$. In other words, the sum iterates over images with different sets of masked superpixels. A function $h_{x_m} : \mathbb{R}^n \to \{0,1\}^L$ divides an image into superpixels and is different for every image $x_m$, therefore, $h_{x_m}(x_m) = q_m$ and $h^{-1}_{x_m}(q_m) = x_m$. As before, $L$ is the number of superpixels the image is divided into. Lastly, $|q'_m|$ stands for the number of non–zero elements in $q'_m$ or, in other words, a number of superpixels that are not masked.

## 2.2. Evaluation metrics

An evaluation metric is an algorithm that gives scores to different explanation methods thus making it possible to compare them. Each metric uses a different technique and therefore captures different characteristics of the evaluated explanation methods.

Table 2.1: Summary of metrics from the Quantus Python library that are used in our work for evaluating explanation methods.

| **Metric name** | **What score is better?** | **Quantus metrics category** |
|---|---|---|
| Average Sensitivity | lower | Robustness |
| Faithfulness Estimate | higher | Faithfulness |
| Iterative Removal of Features | higher | Faithfulness |
| Sparseness | higher | Complexity |

The Quantus [33] framework implements many evaluation metrics divided into 6 categories which can be also interpreted as desired properties of explanation methods. Each category gathers metrics that measure a similar property of attributions generated with explanation methods. Some categories have metrics that can be widely used while others are more specific and have more constraints such as the need for ground truth attribution maps.

**Faithfulness** (called also as Correctness [10]) Assesses the faithfulness of the explanation in relation to the examined model. It measures the correspondence between computed attributions and the actual behaviour of the model. This desired property regards the descriptive accuracy [41] of the explanation method, whether it grasps the relationships in data learned by the model. It is worth highlighting that faithful explanation presents the model's perspective, it does not have to look sensible for a user or follow the human intuition [31], [42]. The ideal explanation should mark features accordingly to their importance for the model's prediction.

**Robustness** These metrics quantify variations of examined method's output with regard to perturbations made in the model's input. They measure if the explanations are stable – slight changes in the input image that do not affect the model's prediction much, should not affect the attribution significantly. This property may be also referred to as Continuity [10] which relates to the continuity of the explanation function.

**Complexity** (also called Compactness [10]) To be user-friendly and comprehensible, the explanation method should not output complex results [43]. The number of features indicated as important should not be excessive. Marking most of the features as relevant could be confusing for a user who tries to understand the model's operation, especially when attribution has similar values. The ideal method should produce concise explanations, highlighting only those features that are most important for the model's prediction.

**Localisation** Localisation metrics quantify whether features designated as most important are situated inside the ground truth region of interest (ROI). They can be used in the explanation method development framework [25]. For our use case of selecting the best method for a given trained model, Localisation metrics should not be used. When most relevant features are outside of ROI, that may come just from the model, and the explanation method has done its duty well [42], [44]. Additionally, due to the need for ground truth attribution maps these metrics are not suited for a tool that attempts to lower the entry threshold for comparing explanation methods.

**Randomisation** These metrics use randomisation tests to assess the adequacy of explanation techniques. Randomisation metrics assess how the output of examined explanation function changes on randomly perturbed input (e.g. model weights, image). Insensitivity to random perturbations of values that the user wishes to examine (e.g. the influence of model's parameters on prediction) could put the usage of such explanation in question [45]. Such metrics can play the role of sanity checks for attribution-based methods.

**Axiomatic measures** This category consists of miscellaneous axiomatic properties such as invariance to input shift [46] or Completeness [10] – fulfilled when the sum of attribution values should equal to the difference of $f(x) - f(x')$, where $x$ is the examined observation and $x'$ is a baseline, e.g. mean values across the dataset.

As presented in Table 2.1, selected metrics are Average Sensitivity [8] from the Robustness category, Faithfulness Estimate [12] and IROF [13] from the Faithfulness category and Sparseness [14] from the Complexity category.

### 2.2.1. Average Sensitivity

Sensitivity [8] in general is a metric that measures how sensitive the explanation method is. It is one of the desirable properties of an explanation method, that insignificant perturbations in input do not change an attribution too much - it should be robust to a certain level. However, the attribution should change visibly when these perturbations change the prediction of the model. If the attribution changes a lot, while the input and model's prediction are more or less stable, it raises doubts, about if the explanation method is suitable for a given problem. Therefore, in general, the smaller the sensitivity level, the better.

However, it is not always the best solution to minimize this measure, not taking other aspects into consideration. Sensitivity looks only at one of the aspects of a good explanation method and is not enough to judge them perfectly. The reason why minimizing sensitivity and not looking at other metrics is not a good approach is simple – optimal explanation, that has the best sensitivity score is a constant explanation. In that case, no matter how the input changes, the explanation method outputs the same result. This kind of explanation method would not be useful to show the model's behaviour. Additionally, some of the explanation methods calculate sensitivity, while creating attributions, which makes it impossible to minimize sensitivity endlessly. A complex solution for those problems is seen in [8], where the sensitivity was one of the metrics and there

was no optimization conducted with respect to the average sensitivity score. Therefore this was not the case for creating the AutoeXplain package. The implementation of Average Sensitivity which is used is based on approximations with the use of Monte Carlo sampling [8].

### 2.2.2. Faithfulness Estimate

Faithfulness Estimate [12] evaluates the relevance of the computed explanation. In a perfect scenario, it would compare computed feature attribution with the ground truth values. However, the ground truth importance of features may be unknown and therefore the paper that introduces this evaluation method also proposes a way of working without these ground truth importances.

Suppose there is an image divided into features and each of the features can be removed, e.g. with a zero value baseline. When some features are removed the model should change the output accordingly to the importance of the removed features. The metric value is the correlation between the importance of the removed feature and the probability drop after removing that feature.

Depending on the nature of input data, sometimes a viable way of removing features may be to change some of the model's coefficients to 0. However, in the case of a CNN model and images as input, the proposed way of removing features is by changing color of a group of pixels to black or to their average color.

### 2.2.3. Iterative Removal of Features

Iterative Removal of Features (IROF) [13] measures the influence that removal of the most important image parts (according to the evaluated explanation method) has on the predicted score for a given class. Its major advantage is a low computational cost compared to other faithfulness metrics.

Consider the following setup: let $x_m \in \mathbb{R}^n$ be a m–th image, segmented into set of $L$ disjoint segments $\{S_m^l\}_{l=1}^L$, e.g. using the S.L.I.C. segmentation algorithm as proposed in [47]. Let $F : \mathbb{R}^n \to \mathbb{R}^k$ be a neural network for k–class classification problem and $v_m \in \mathbb{R}^n$ stand for the computed attribution for $x_m$. For each segment, a mean relevance is computed using

$$\frac{\left\| v_m S_m^l \right\|_1}{\left\| S_m^l \right\|_1}, \tag{2.11}$$

with $S_{m,i}^l = 1$ indicating that pixel i–th pixel of m–th image belongs to segment $l$. This makes it possible to compare two segments. Next, segments are sorted by mean relevance in descending order, regardless of the segments' size. Let $w_m^l$ be image $x_m$ with $l$ most relevant segments replaced with the mean value of pixel (across the whole dataset). Computing $F^c\left(w_m^l\right)$ for each $l \in 0, ..., L$ gives a curve for the score for some class $c$ against the number of removed segments.

A good explanation will assign high relevance to areas of the image that are significant for predicting the given class. Because segments of the highest relevance are removed at first, the score for a given class will drop rapidly, increasing the area over the curve (AOC). IROF score calculated for some explanation method $V$ averaged across all image examples $x_m$ for $m \in 1, ..., M$ is calculated with:

$$\text{IROF}\,(V) = \frac{1}{M}\sum_{m=1}^{M} \text{AOC}\left(\frac{F^c\left(w_m^l\right)}{F^c\left(w_m^0\right)}\right)_{l=0}^{L}. \tag{2.12}$$

### 2.2.4. Sparseness

Sparseness [14], with the use of Gini Index [48], measures the impact of irrelevant features on the explanation methods attribution. The advantage of this metric is that it evaluates if the explanation method generates attributions that are concise and highlight only features with the most impact on the model's prediction. The values of the metric are between 0 and 1, where 0 means the attributions are not sparse and 1 means that features marked as irrelevant had no impact on the final model's decision; thus, higher values indicate better results.

The metric shows an impact of features on the attribution, not taking into account if the value was above or below 0. That is why the absolute value of all values for attribution is calculated. Therefore, for m–th image $x_m \in \mathbb{R}^n$, consider sorted (in non-decreasing order) attribution vector $v_m$ of length n: $v_m = [v_{m,1}, ..., v_{m,n}]$, containing non-negative values.

Value $v_{m,i}$ is i–th value of this vector $v_m$. The Gini Index (which directly gives the sparseness value) is calculated with formulae:

$$G(v_m) = 1 - 2\sum_{i=1}^{n} \frac{v_{m,i}}{||v_m||}\left(\frac{n - i + 0.5}{n}\right). \tag{2.13}$$

For explanation methods, the $G(v_m)$ is calculated for some input vector $v_m$. The higher the value of $G(v_m)$, the more sparse an attribution.

# 3. AutoeXplain Python package

The AutoeXplain Python package allows users to automatically select the best explanation method for a given problem. To get the results, the user has to provide two types of input data: CNN model and image dataset. The final results of a package are information on a selected explanation method and reports in both HTML and PDF format.

The package selects the best XAI method from four possible methods: Grad-CAM, Integrated Gradients, KernelSHAP and Saliency. The selection process is done based on aggregating the results of the evaluation of those methods according to four metrics: Average Sensitivity, Faithfulness Estimate, IROF and Sparseness.

We opened a source code of package implementation in the GitHub repository.[1] Further in this thesis, we describe the technology used in this package, more detailed information about the package and implementation.

## 3.1. Technology selection

Our solution is based on PyTorch [49] library version `1.12.1`. It is a tensor computation Python library, with automatic differentiation functionality. One of the common use of PyTorch is building neural networks. In contrast to other deep learning solutions, it uses a dynamic computation graph, which simplifies prototyping and allows writing code in a more *Pythonic* manner. It supports CUDA GPU acceleration, which gives substantial speed-up of computation time. The next key library is Captum [50] version `0.5.0`. It implements numerous attribution-based explanation methods, designed to work with PyTorch models, therefore our solution can easily work with these explanation methods. The vital mechanics of our solution is evaluating explanations with a series of evaluation metrics. For this purpose, we use the Quantus [33] library which implements a set of evaluation metrics designed for attribution-based explanation methods used on neural networks. We use version `0.1.6` of the Quantus library. For generating HTML reports, our solution utilizes `Jinja2` template engine. To generate PDF reports, `MiKTeX` typesetting system is used together with `PyLaTeX` Python library.

[1] Source code of the package can be found in the GitHub repository available at https://github.com/MI2DataLab/autoexplainer.

## 3.2. Package description

When initializing the package, the user can choose to run it on CPU or CUDA. Additionally, the user has to provide data and a model. After that, the main function of the package `evaluate()` can be run. It calculates the results of explanation methods for given input data. Users can choose, whether to run all four methods or only some of them. For each of the images from the dataset, attribution (a result of the explanation method) is created. Later, each of the attributions is evaluated by metrics. Also, in this case, the user can choose to calculate results only for a subset of four available metrics. After evaluating all of the attributions, the results are aggregated. The details about this process can be found in Section 3.3. Based on the aggregation results, the best explanation method can be selected. The user can get both the name of the selected method and its parameters. It is also possible to generate reports both in HTML and PDF format. The report contains a summary of the selection process. It includes partially aggregated results of the evaluation, details about the selected method and examples of images from the dataset with attributions. A more detailed description of the reports is in Section 3.3.6.

For now, the package includes four XAI solutions and four metrics to evaluate them, however, it is implemented in such a way that it is possible to add new explanation methods and metrics in the future.

## 3.3. Implementation details

This section describes details of the major mechanisms in our solution. During package development, we found and fixed a bug in the Quantus library, which has to be fixed for further use of it. More details are presented in Section 3.3.4. Baseline settings for explanation methods and evaluation metrics are presented in Section 3.3.5.

### 3.3.1. AutoExplainer evaluation and selection

`AutoExplainer.evaluate` method performs an evaluation for provided set of explanation methods and metrics. As metric values are computed for each image for each explanation method for each metric, two steps of aggregation are done to obtain a comparable single-valued score for each method. To address the problem with different-scaled metric values – as they cannot be easily compared with each other – we implemented ranked-based scoring, a variation of Borda count. For each used metric, evaluated explanation methods are granted points based on their ranking position with respect to given metric values. This approach also solves the problem with different monotonicities of metrics – when for one subset of metrics, higher values indicate a higher score, and for another subset of metrics, lower values indicate a higher score. In the case of a tie in top-scored methods, the method is selected based on its computation time – shorter is better.

### 3.3.2. Explanation handlers

`ExplanationHandler` is an abstract class designed to manage the functionality of explanation methods. Each method has a dedicated class that inherits from `ExplanationHandler`, e.g. `SaliencyHandler`. During initialization, these objects set the method's parameters based on the user's explicit input or default values based on passed model and data, e.g. extracting the last convolution of the network layer for the Grad-CAM method when the layer is not selected by the user (what is recommended [7]). To evaluate methods, Quantus metrics need an explanation function object to be passed. This function needs to fulfil a particular interface defined by the Quantus library. `ExplanationHandler`, after setting the parameters, it creates a function object tailored for this interface.

Thanks to these mechanics in Explanation Handler classes, any local attribution-based explanation method can be evaluated with Quantus metrics – if the results of the given explanation method is a heatmap of the shape of an input image. This implementation makes AutoeXplain easily extendable with new explanation techniques.

### 3.3.3. Metric handler

`MetricHandler` is an analogous abstract class as `ExplanationHandler` but for evaluation metrics. The main functionality of metric value computation is implemented in the base class. Only parameter handling is done in implementations dedicated to concrete metrics.

### 3.3.4. Bug fixes in the Quantus library

During our work, we came across a bug in the Iterative Removal of Features metric implementation in the Quantus library. During IROF computation, perturbation of subsequent segments should be accumulated. However, in its implementation, segments were perturbed one by one. Hence, incorrect AOC values were returned. We contributed a fix to the mentioned problem in the pull request to the Quantus repository.[2]

### 3.3.5. Baseline settings

Each evaluation metric and explanation method has a collection of parameters to be set. Generally, we followed the settings recommended by the authors of a given solution. However, some parameters have a direct impact on execution time. Since attribution functions are run repeatedly for each provided image during their evaluation, for such parameters we set lower settings than recommended (in terms of computation time).

**Grad-CAM** This method needs a convolutional layer to be selected for its attribution computation. As recommended [7], the last `torch.nn.Conv2D` layer of the network is selected. To match the original Grad-CAM algorithm, ReLU activation is applied to the result of the explanation function.

[2]Pull Request is available at https://github.com/understandable-machine-intelligence-lab/Quantus/pull/221.

**Integrated Gradients** The baseline image color is set to black (zero-values tensor) as recommended [6]. The default number of steps used by the approximation method is set to 20.

**KernelSHAP** The baseline image color is set to black (zero-values tensor). The default number of samples used to train a LIME surrogate model is set to 50. As KernelSHAP needs superpixels for attribution computation, the S.L.I.C. [47] segmentation algorithm is used by default.

**Saliency** Saliency method has no parameters.

**Average Sensitivity** Average Sensitivity is computed on 20 samples by default.

**Faithfulness Estimate** The baseline image color is set to black (zero-values tensor).

**Iterative Removal of Features** The baseline image color is set to the mean color value across the whole dataset, which is expected to give better results than black color [13]. The S.L.I.C. [47] segmentation algorithm is used by default as in IROF paper [13].

**Sparseness** Sparsensess metric has no parameters.

### 3.3.6. Reports

One of the contributions of this thesis is generating both HTML and PDF reports, containing details about the evaluation process of XAI solutions for a given problem. The AutoeXplain package implements two functions for generating reports: `AutoExplainer.to_html` and `AutoExplainer.to_pdf`. As mentioned before, while creating reports in `.html` format, we use `Jinja2` template engine. To generate a PDF report it is necessary to have `LaTeX` installed, as it is used by `PyLaTeX` Python library. Both reports contain the same information, examples of them can be found in Appendix A and Appendix B. In this section, we will discuss the report in detail using the example of the HTML version.

There are seven important parts of this report: General information, Model performance, Table of results, Examples of computed attributions, Aggregation parameters, Explanations, and Metrics.

**General information** We extracted the base information about the conducted selection to print at the beginning of the reports as on Figure 3.1. They include:

- Date, when the report was generated.
- AutoeXplain package version, with which the report was generated.
- Number of images provided by the user.
- Time spent on running the whole selection process of the best explanation method.
- Name of the selected method.

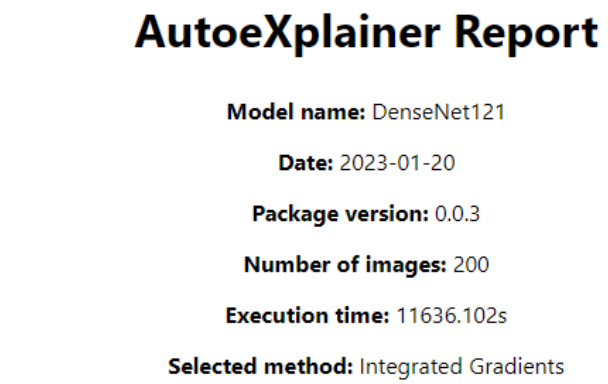



Figure 3.1: The beginning of the HTML report.

**Model performance** After basic information, the report provides information about the performance of the model, because it is a significant factor in explanation method's operation. As in Figure 3.2, there are three measures of the model's performance in this section: Accuracy, F1 macro score and balanced accuracy score (BAC), all implemented in scikit-learn Python library [51]. Accuracy focuses on counting the correctly classified labels and returns the fraction of correctly classified labels. The F1 score is calculated as the harmonic mean of precision and recall. We use the F1 macro version of this metric, which means that the F1 score is computed for every class and averaged across them. A balanced accuracy score is defined as the average recall for each class. F1 macro and BAC are suitable metrics for imbalanced datasets.

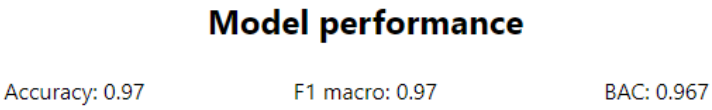



Figure 3.2: Section about model performance.

**Table of results** In the next section of the report, there is a table with the results of the evaluation and selection process as in Figure 3.3. In rows, there are names of the explanation methods that were evaluated, and in columns, there are names of metrics described in Section 2.2. The arrows next to metric names indicate the monotonicity of metric – larger (↑) or smaller values are better (↓). Numbers in the table represent the mean value of the metric for a given explanation method. *Time elapsed* compares the execution time of explanation methods for the provided batch of data. The results of the aggregation algorithm for each method are in column *Agg. Score*. The aggregation algorithm is shortly described in Section 3.3.1. Explanation methods are sorted based on aggregation score.

**Table of results**

| Explanation Name | Rank | Faithfulness Est. ↑ | Avg Sensitivity ↓ | IROF ↑ | Sparseness ↑ | Time elapsed [s] | Agg. Score |
|---|---|---|---|---|---|---|---|
| **Saliency** | 1 | 0.071 | 0.063 | 36.692 | 0.588 | 9.771 | **8** |
| **GradCam** | 2 | 0.245 | 0.142 | 42.318 | 0.377 | 4.082 | **7** |
| **Integrated Gradients** | 3 | -0.144 | 0.049 | 31.993 | 0.582 | 12.214 | **5** |
| **KernelSHAP** | 4 | 0.025 | 0.306 | 37.367 | 0.503 | 179.245 | **4** |

Figure 3.3: Section with results of the selection process.

**Examples of computed attributions** The report includes sample visualizations of attribution maps to allow the user to familiarize himself with the explanations and to compare results between classes. There are $min(10, number\_of\_images\_in\_dataset)$ images printed. If the number of classes is smaller than the number of printed images, multiple images from each class are chosen in a balanced way. Figure 3.4 shows the first 3 rows of the whole 10–rows image.

For each sample image, the report presents the true class of the image, the class predicted by the model and its predictions score, which indicates the certainty of classification. On the right from the original images, there are images with attributions created by consecutive explanation methods. In general, the green parts on those images are the parts of the image, which model considered to have a positive impact on predicting chosen class (positive attribution values), red parts – to have a negative impact (negative attribution values). White color corresponds to zero attribution value. A more detailed explanation is described in Section 2.1.

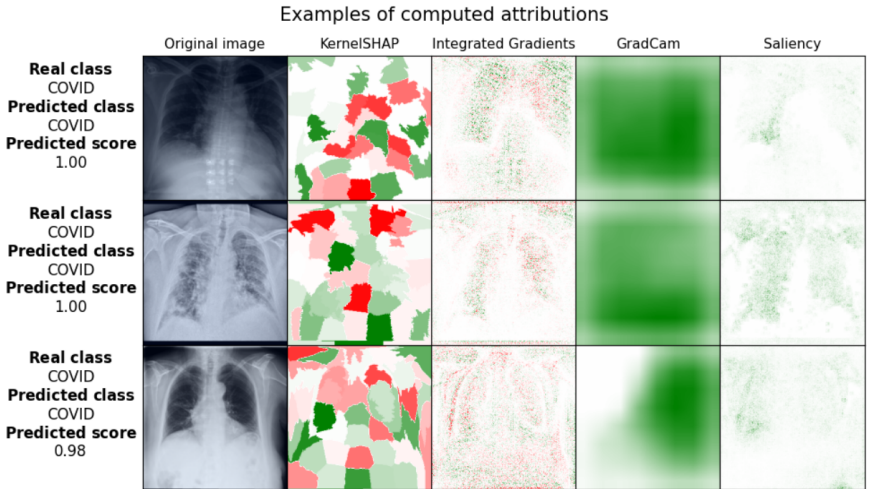



Figure 3.4: Section with examples of images from a given dataset with corresponding calculated attributions.

**Aggregation parameters** Each method is assessed by each metric for every image separately. That is why there was a need to aggregate those results, to compare the methods between them. The aggregation process has two stages in the AutoeXplain package. The first one computes the general score for each method and each metric, by default by calculating the mean of the

values for all images from the dataset. Next, the second stage aggregates the results for each explanation method, reducing them to the form of a single value per method, allowing to order them in terms of aggregated score. Aggregation stages can be done in a different way than by default described in Section 3.3.1. In order to make the selection process reproducible, aggregate parameters are included in the report as shown in Figure 3.5.

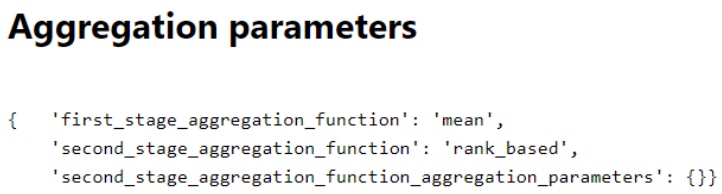



Figure 3.5: Report section containing aggregation parameters.

**Explanations** For each used explanation method, there is a short description in this report section as shown in Figure 3.6. The description includes short summary of the idea of the method and a link to the original paper describing it. Additionally, as some of the methods' parameters can be set in the AutoeXplain package, the most important ones are also printed in this section. It is important to mention, that the report includes a description only of used methods, methods unused in an experiment are not shown. To make the selection process reproducible or allow users to use a particular method with implementation from a different library, we included parameters set for each explanation method. If some parameter is not printed, the default value determined in the Captum library implementation is used.

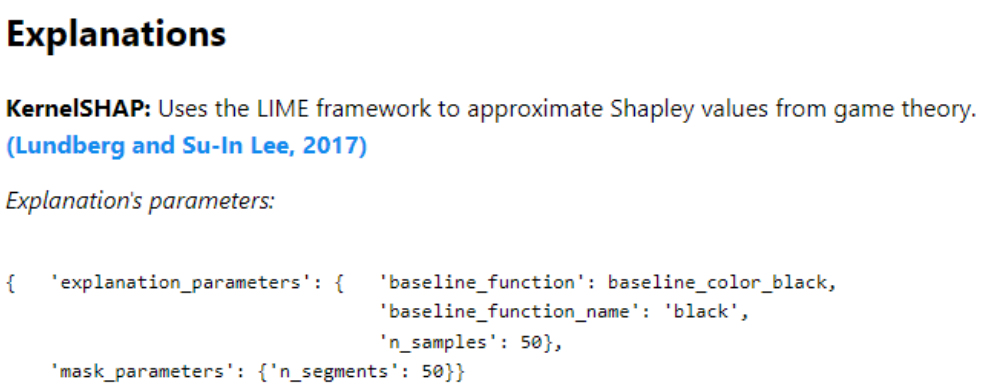



Figure 3.6: Section about parameters for explanation methods.

**Metrics** Similarly to explanation methods, this section includes the descriptions for metrics. We can find the short summary of the metric, a link to its article and non-default parameters set for evaluation, as shown in Figure 3.7.

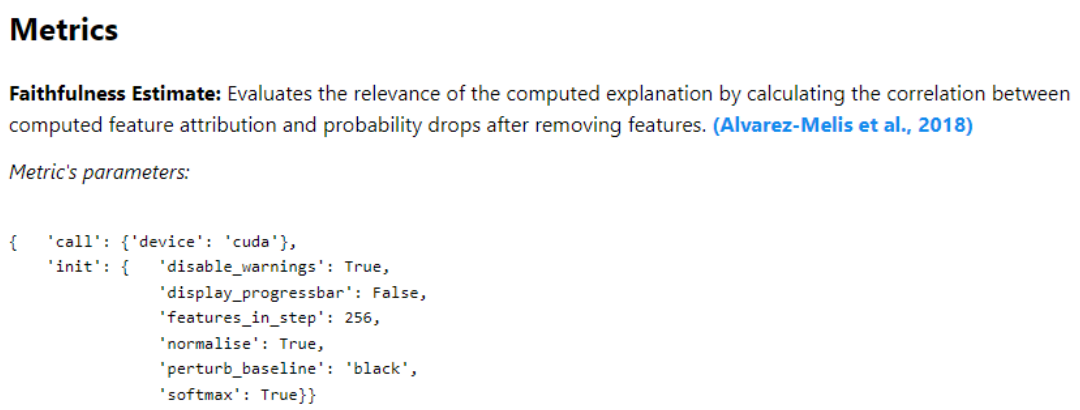



Figure 3.7: Section about parameters set for each metric.

## 3.4. Example of usage

Below there are examples of how to use (in Python language script) package to find the best explanation method for a given model and dataset. First Listing 3.1 shows the simplest usage, with only four lines of code the user can obtain the selected method for his problem. Second Listing 3.2 shows a more advanced example. A subset of evaluated methods is selected and their parameters are set. The selected explanation method can be used to explain other data or to be evaluated on it. Raw information about the evaluation and selection process can be accessed via attributes of `AutoExplainer` object.

Listing 3.1: Simple example of AutoeXplainer usage.

```
from autoexplainer.autoexplainer import AutoExplainer

model: torch.nn.Module
data: torch.Tensor
targets: torch.Tensor

explainer = AutoExplainer(model, data, targets, device='cuda')

# Evaluate all available methods with all available metrics.
explainer.evaluate()

# Aggregate results.
explainer.aggregate()
```

```
# Obtain an object for the selected explanation method.
explanation_method = explainer.get_best_explanation()

# Get attribution maps for provided images.
explanation_method.attributions

# Generate report in HTML format.
explainer.to_html("report.html")
```

Listing 3.2: Advanced example of AutoeXplainer usage.

```
from autoexplainer.autoexplainer import AutoExplainer

model: torch.nn.Module
data: torch.Tensor
targets: torch.Tensor

explainer = AutoExplainer(model, data, targets, device="cuda")

# Evaluate a subset of methods with a subset of metrics. Set parameters for
    explanation and metrics.
explainer.evaluate(
    explanations=["integrated_gradients", "kernel_shap", "saliency"],
    metrics=["average_sensitivity", "irof", "sparseness"],
    explanation_params={
        "kernel_shap": {
            "explanation_parameters": {"n_samples": 10},
            "mask_parameters": {"n_segments": 10},
        },
    },
)

# User can get relevant information from attributes.
explainer.raw_results # all metrics results for all images
explainer.times_methods
explainer.times_metrics

# Aggregate metric scores and see aggregated results.
explainer.aggregate()
explainer.first_aggregation_results # results aggregated into single value per
    (method, metric) pair
explainer.second_aggregation_results # results aggregated into single value per method

# Produce HTML and PDF reports.
explainer.to_pdf(
    "report.pdf", model_name="DenseNet", labels={0: "dog", 1: "cat", 2: "fish"}
)
```

```
# Get the selected explanation method for later use.
explanation_method = explainer.get_best_explanation()
attributions = explanation_method.explain(model, other_data, other_targets)

# Evaluate the explanation method on other data.
explanation_method.evaluate(model, other_data, other_targets)
```

### 3.5. Possible scenarios of usage

The AutoeXplain package can be used by various users interested in explaining their model. Users with no advanced knowledge in the XAI area can obtain a decent explanation method for their model and data. Its selection will be backed by objective and theoretic-grounded criteria. For advanced users who carry out a search for an explanation method for a particular model and data, this package can be a beneficial aid for the research process. Automatic selection can serve as a good baseline, and the evaluation report can be a starting point for further search. The AutoeXplain package can be applied during model development too. During carrying out experiments with different versions of a model, the results of method evaluation can be used as another experiment artefact. Moreover, generated report can be attached to the model's documentation describing its explainability aspects.

# 4. Experiments

To show the results that users can get from the AutoeXplain package we conducted experiments for three different input pairs – dataset and model. This section shows more details about chosen datasets and models trained on them. Later we provide a short description of the devices used to perform the tests. At the end of this chapter, we provide sample results of those experiments, based on the reports generated by this package. Code for generating reports, reports themselves, and used models' weights are placed in the GitHub repository.

## 4.1. Experimental setting

The AutoeXplain package needs both a dataset and a model as input in order to evaluate explanation methods. For the experiments, we used 3 different datasets containing various types of data: synthetic images, X-ray scans and colorful photos.

### 4.1.1. Datasets

Datasets, for which the models were trained, were selected to be very different from each other: one dataset contains grayscale X-ray scans, another contains colorful photos and the third one consists of simple geometric shapes on grey background. Sample images from all datasets are presented in Figure 4.1. Datasets that were used, are balanced – for each dataset, its classes have a similar number of images. At the time of writing, all data was publicly available. The datasets used are:

- **Chest X-ray** [16] – a medical dataset with 3 classes and 33920 gray scale Chest X-ray scans. The classes show the lungs of healthy patients, the lungs of patients that have COVID-19, and the lungs of patients that are ill with something else.
- **Imagenette** [17] – a subset of the ImageNet dataset [52]. It is limited to a bit over 13000 colorful photos from 10 classes: tench (a fish), English springer (a dog), cassette player, chain saw, church, French horn (a musical instrument), garbage truck, gas pump, golf ball and a parachute. In the version of Imagenette used here, all images had their shortest side resized to 320 pixels while maintaining their aspect ratio.
- **Kandinsky Patterns** [15] – this paper introduced 3 datasets and the one used here is called Challenge 1. It is a synthetic dataset of 3000 images equally divided into 3 classes. Every image contains many small geometrical shapes that are either red, blue or yellow

and they can either be a circle, triangle or square. All are placed on a grey background. Classes are differentiated by patterns that these shapes are placed into – as an example, a class that is called true, has a set of rules that created patterns must match: small shapes are placed in such a way that they create 2 bigger, different figures. Furthermore, each big figure must be created using small shapes that are of specific colors and shapes other than the big one. For example, a big square is built using small circles and triangles that are either red or blue.

The Chest X-ray dataset has already been split into training, validation and test sets with 21715, 5417 and 6788 images respectively. Later, during models training, we removed the validation set and used the test set in its place. This reduced the number of images from 33920 to 28503.

Similarly, the Imagenette dataset has also been split. However, it has been split into 2 sets: training and validation with a ratio of 70 to 30.

On the other hand, the synthetic dataset Kandinsky Patterns is not split by default. Therefore, it was split by us into training and validation sets with a ratio of 90 to 10.

(a) (b) (c)

Figure 4.1: Sample images from the datasets used for tests: (a) Chest X-ray, (b) Imagenette and (c) Kandinsky Patterns. Images are from classes: (a) lungs of a patient with COVID-19, (b) English springer, (c) true – pattern that does match all rules.

### 4.1.2. Models

The AutoeXplain package is intended for CNN models from the PyTorch [49] library therefore models used in the experiments are `torch.nn.Module` objects. First, for each of these 3 datasets, we trained models with different settings, to choose the best one for further work. We trained models with:

1. architecture of DenseNet121, DenseNet161, DenseNet169 and DenseNet201 [16],
2. either pre-trained on ImageNet dataset or not.

Other parameters set were: number of epochs was 5, batch size - 32 and learning rate - 0.0001. Also, images were resized to a common size of 256 x 256 pixels and normalized in the same way

as images from the Imagenet. This normalization should help pre-trained models as they were trained on normalized images from the ImageNet dataset.

Next, models were compared by their accuracy on the training and validation sets. Finally, for each dataset, a suitable model was selected with scores presented in Table 4.1. All of the selected models share the same architecture of DenseNet121 as more complex DenseNet architectures did not have much effect on scores.

Table 4.1: Scores of the models that were trained and then used in the experiments. Since Imagenette is a subset of the ImageNet dataset and the model was pretrained on the latter, the model's accuracy for Imagenette is very high.

| **Dataset** | **Model architecture** | **Accuracy train set** | **Accuracy validation set** | **Pretrained** |
|---|---|---|---|---|
| Chest X-ray | DenseNet121 | 0.933 | 0.925 | No |
| Imagenette | DenseNet121 | 0.998 | 0.972 | Yes |
| Kandinsky Patterns | DenseNet121 | 0.950 | 0.893 | No |

The goal of training these models was to achieve high accuracy so that when the attributions are computed, the model would usually predict the correct class.

## 4.2. Execution plan

To show the work of created package, we used models and data described in Table 4.1. Due to the fact, that functions of the package were running slow on our personal computers on CPU only (on average 7.5 minutes per image for 4 explanation method and 4 evaluation metrics), we decided to introduce GPU acceleration. It helped, as the computation of attributions for one image with the same setting took only about 1.5 minutes with the use of NVIDIA GeForce RTX 2060 with 6GB VRAM. Still, to conduct more experiments faster, on bigger sets of images, we decided to use High Performance Computing (HPC) cluster. The HPC cluster that was used, is located in the MiNI Faculty with Ubuntu 20.04.5 LTS system. It contains various resources, including 32 Graphical units NVIDIA A100, with 40GB VRAM each, which we used for the experiments.

Models were trained on whole datasets, but for experiments with the use of the AutoeXplain package, we randomly selected a subset of images from each dataset. To compare execution times on a personal computer and HPC, we run an experiment on 10 images. To compare execution times with and without GPU, experiments were run once using 50 images and on HPC only. For other experiments, a subset of 200 images was used.

## 4.3. Results

Below we present the results of the experiments conducted during the time, we worked on the AutoeXplain package. First, we compare the execution times of a selection process computed on a personal computer and on HPC. Secondly, we analyze the results of the conducted experiments. In Section 4.3.3 we summarize them and draw conclusions, useful for future uses of the package.

### 4.3.1. Comparison of execution times

The results described in this section are based on the experiments conducted for a subset of the Chest X-ray dataset.

**Comparison between personal computer and HPC cluster** As described in Section 4.2, we run part of the experiments both on a personal computer and on the HPC cluster. As foreseen, the differences between execution times on CPU and GPU are easily visible. Also, the differences between our computer and HPC are worth mentioning. For our computer on CPU time to run the whole process of selecting the best method for 10 images from Chest X-ray took about 73.8 minutes. For GPU it was nearly 5 times faster. On HPC the sample of also 10 images from the same dataset, took about 33.6 minutes on the CPU and 10 minutes on the GPU. The summary for each of the settings is shown in Figure 4.2. The faster one in these experiments was HPC.

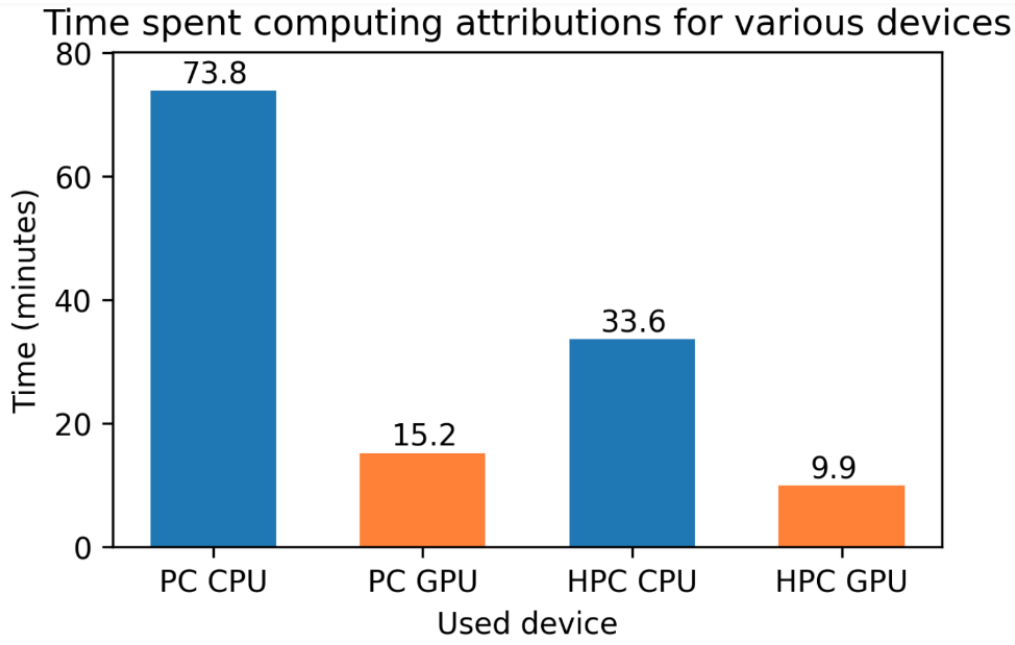



Figure 4.2: Bar plot with the comparison of execution times for various settings for 10 images.

**Detailed comparison between CPU and GPU on HPC** Moreover, we decided to compare the average time, it took to compute the results for CPU and GPU on HPC. As expected, the times are different. The computations with the use of GPU took about 3 to 4 times less time than with CPU-only, as shown in Table 4.2. The total times for each dataset are similar, as in all three cases the models used had the same architecture (all were DenseNet121).

Table 4.2: Average time needed to analyze 1 image using AutoeXplain with and without GPU. The last column is a ratio of CPU time to GPU time. Values are computed by dividing the total execution time by 50 images. These results are based on a single execution of analysis for every dataset with default parameters.

| Dataset name | CPU time [s] | GPU time [s] | CPU time / GPU time ratio |
|---|---|---|---|
| Chest X-ray | 207.1 | 60.1 | 3.4 |
| Imagenette | 250.9 | 59.0 | 4.3 |
| Kandinsky Patterns | 202.2 | 58.5 | 3.5 |

Additionally, we compared the execution times for each of the metrics and methods separately. The data was also collected during the experiments run on GPU on HPC. In Figures 4.3 and 4.4 we can see the results. When it comes to methods available in the AutoeXplain package, clearly, the slowest is KernelSHAP. The other three methods seem to be similarly fast – time spent on computing attribution for one image is for all of them under 1 second. For metrics available in the package, the fastest is Sparseness. For the experiment on 200 images chosen from Chest X-ray dataset with GPU, computing Sparseness score for one image was on average 7 times faster than the IROF metric, 16 times faster than Faithfulness Estimate and over 28 times faster than Average Sensitivity.

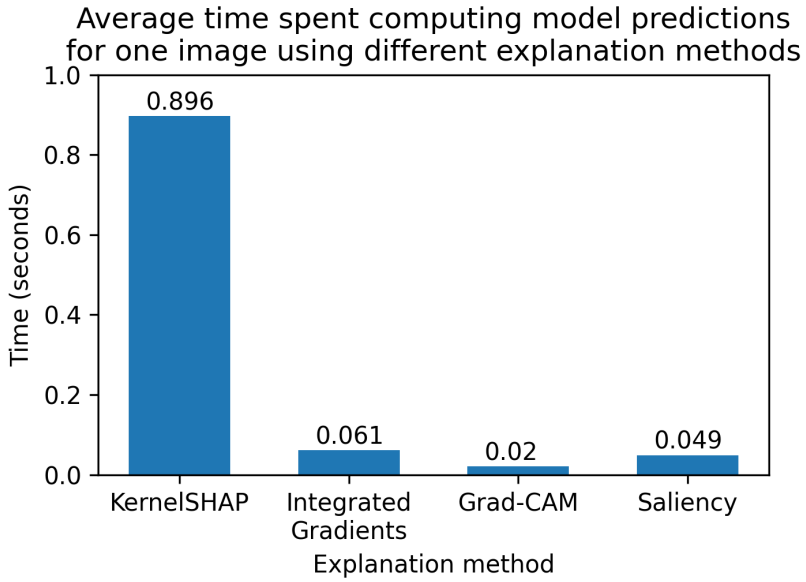



Figure 4.3: Bar plot with the comparison of execution times for all methods available in the AutoeXplain package. The experiment was conducted on HPC cluster with the use of GPU.

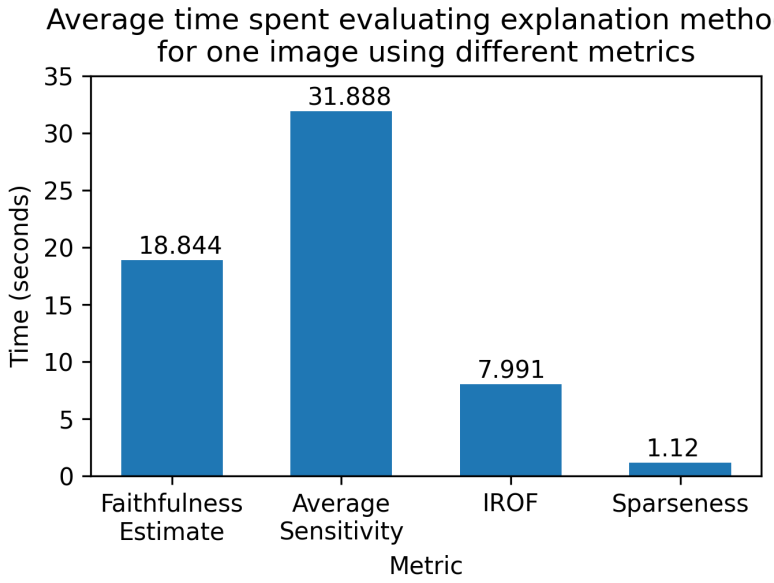



Figure 4.4: Bar plot with the comparison of execution times for all metrics available in the AutoeXplain package. The experiment was conducted on HPC with the use of GPU.

### 4.3.2. Comparison of best methods

In this section, we provide the results of experiments conducted on each of the datasets: Chest X-ray, Imagenette and Kandinsky Patterns. Images in this section are copied from HTML reports generated after the selection process for each of the problems. The experiments were run on the HPC cluster and were using GPU. For each dataset, 200 randomly chosen images were analyzed.

**Chest X-ray** For this dataset the best explanation method, according to the selection process conducted by the AutoeXplain package, is Saliency. As can be seen in Figure 4.5, this method was the best according to metrics: Faithfulness Estimate and Sparseness. For Average Sensitivity and IROF metrics, it was in second and third place, respectively. The examples of the images and the attributions computed with various methods can be seen in Figure 4.6.

**Table of results**

| Explanation Name | Rank | Faithfulness Est. ↑ | Avg Sensitivity ↓ | IROF ↑ | Sparseness ↑ | Time elapsed [s] | Agg. Score |
|---|---|---|---|---|---|---|---|
| **Saliency** | 1 | 0.071 | 0.063 | 36.692 | 0.588 | 9.771 | **8** |
| **GradCam** | 2 | 0.245 | 0.142 | 42.318 | 0.377 | 4.082 | **7** |
| **Integrated Gradients** | 3 | -0.144 | 0.049 | 31.993 | 0.582 | 12.214 | **5** |
| **KernelSHAP** | 4 | 0.025 | 0.306 | 37.367 | 0.503 | 179.245 | **4** |

Figure 4.5: Table with the results of the experiment on the Chest X-ray dataset.

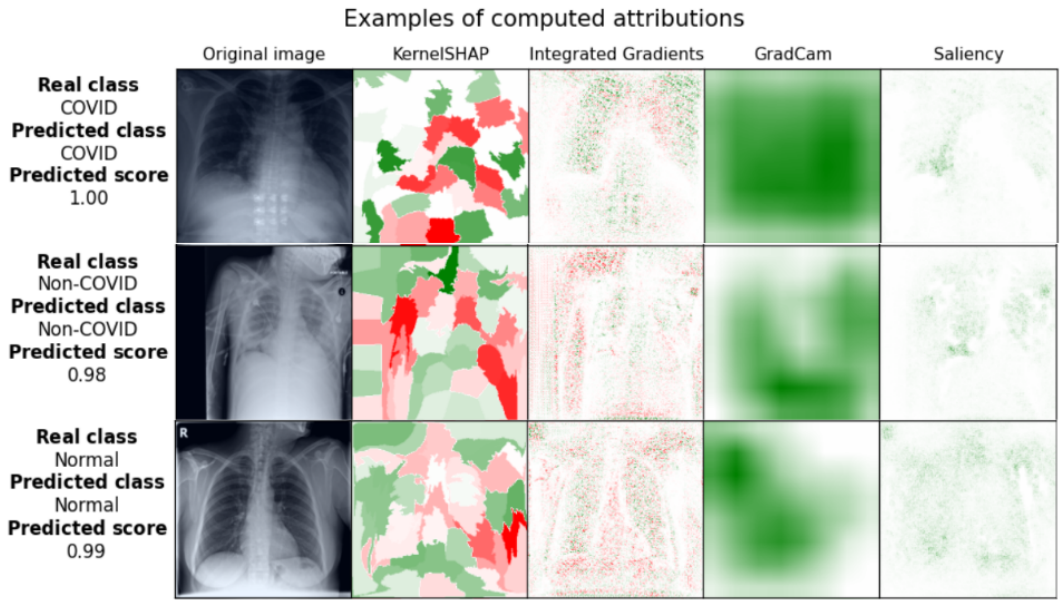



Figure 4.6: Samples of images and attributions created for them from the experiment on the Chest X-ray dataset.

**Imagenette** For this dataset, AutoeXplain selected Grad-CAM explanation method based on 4 metrics. Scores for each metric are in Figure 4.7. Images in this dataset are colorful and they are photos, contrary to X-ray scans in the previous experiment. Not only the input images are different, as can be seen in Figure 4.8, but the selected explanation method too.

**Table of results**

| Explanation Name | Rank | Faithfulness Est. ↑ | Avg Sensitivity ↓ | IROF ↑ | Sparseness ↑ | Time elapsed [s] | Agg. Score |
|---|---|---|---|---|---|---|---|
| **GradCam** | 1 | 0.308 | 0.030 | 47.035 | 0.425 | 4.053 | **9** |
| **Saliency** | 2 | 0.209 | 0.055 | 36.988 | 0.546 | 7.751 | **6** |
| **Integrated Gradients** | 3 | -0.123 | 0.039 | 31.420 | 0.629 | 10.128 | **5** |
| **KernelSHAP** | 4 | 0.085 | 0.296 | 37.847 | 0.442 | 179.301 | **4** |

Figure 4.7: Table with the results of the experiment on the Imagenette dataset.

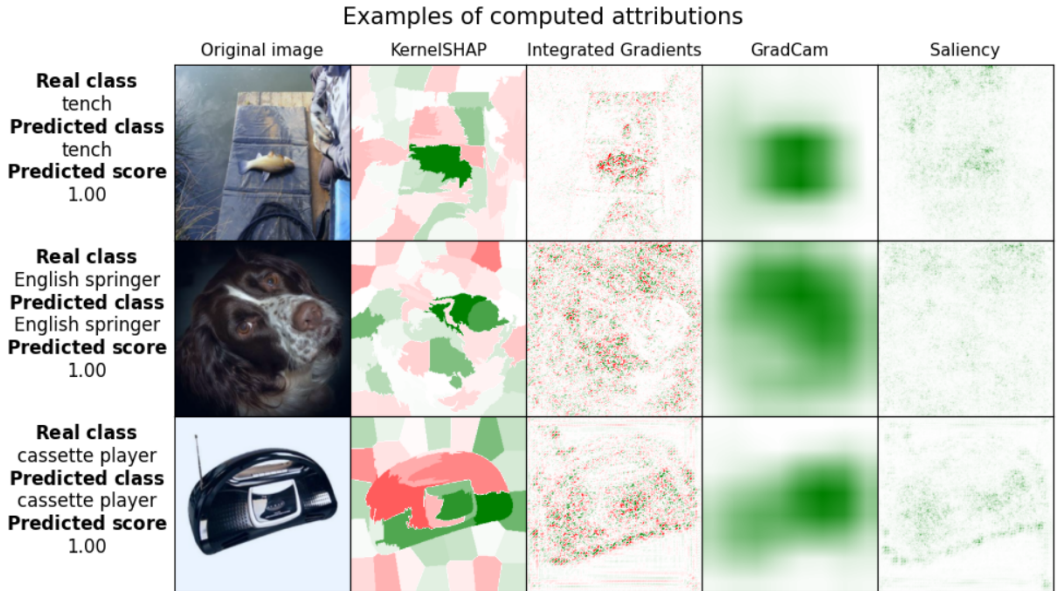



Figure 4.8: Samples of images and attributions created for them from the experiment on the Imagenette dataset.

**Kandinsky Patterns** For the last experiment, the dataset contains simple images of geometrical shapes. This task is different from the previous ones, as every image has a uniform grey background color and does not have any tiny details but clearly visible shapes instead. This time, the AutoeXplain package selected Integrated Gradients as presented in Figure 4.9. The examples of the images with attributions can be seen in Figure 4.10.

**Table of results**

| Explanation Name | Rank | Faithfulness Est. ↑ | Avg Sensitivity ↓ | IROF ↑ | Sparseness ↑ | Time elapsed [s] | Agg. Score |
|---|---|---|---|---|---|---|---|
| **Integrated Gradients** | 1 | 0.103 | 0.054 | 25.415 | 0.550 | 10.119 | **10** |
| **Saliency** | 2 | -0.082 | 0.125 | 25.066 | 0.888 | 7.691 | **6** |
| **GradCam** | 3 | -0.007 | 0.373 | 38.916 | 0.213 | 4.049 | **5** |
| **KernelSHAP** | 4 | 0.023 | 0.484 | 21.099 | 0.473 | 180.731 | **3** |

Figure 4.9: Table with the results of the experiment on the Kandinsky Patterns dataset.

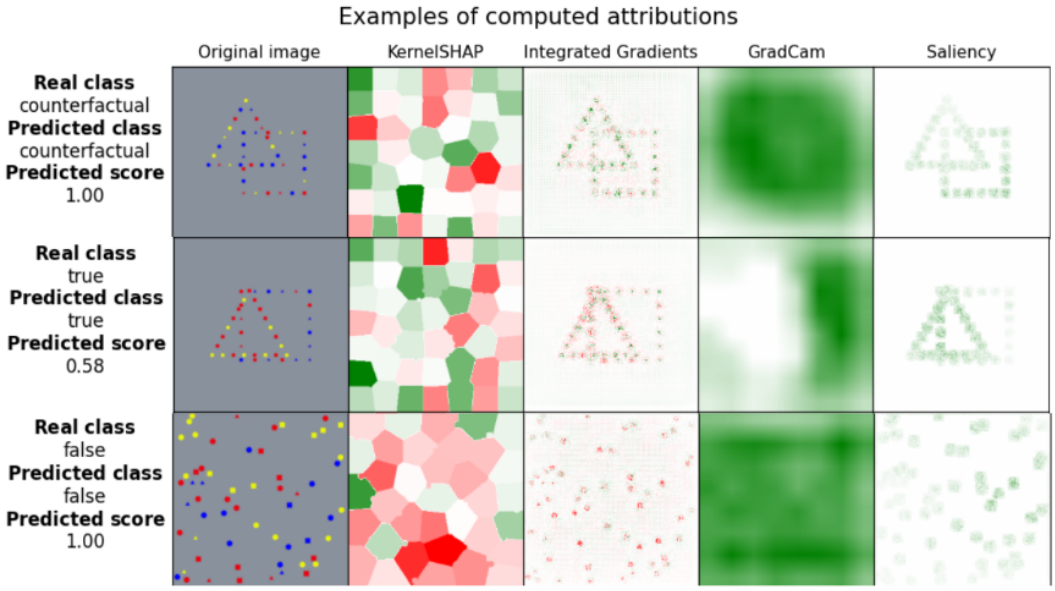



Figure 4.10: Samples of images and attributions created for them from the experiment on the Kandinsky Patterns dataset.

### 4.3.3. Conlusions

As expected, performance gains due to GPU acceleration are significant. Execution of experiments on GPU was done 3-4 times faster than those executed on CPU. The difference between PC GPU and HPC GPU is not as significant - execution on HPC was only 1.5 times faster. Hence, the main advantage of HPC is mainly amount of VRAM in GPU, whereby more images can be used for evaluation and selection. Additionally, more experiments on HPC can be conducted during the same time. For each model and dataset pair, different explanation method has been selected. This demonstrates that each problem needs its own explanation method, even when the architecture of the used model is the same. In each experiment, KernelSHAP were assessed with the lowest score, despite the fact it needed 18-35 times more computation time than the rest of the explanation methods.

# 5. Summary

The created AutoeXplain package is a solution to the problem of selecting explanation methods for CNNs for image data. The package automatically selects the best method for the provided task. No sophisticated knowledge is necessary to use this package, as it is enough to write four lines of code to run the benchmark. After that decent attributions suited for the model and data are created and rated with metrics. The created solution allows the generation of both HTML and PDF reports, providing details about the selection process. These reports can be used also as additional information for the model's documentation about its performance and the possibility to explain it with XAI solutions. The AutoeXplain package can be a beneficial aid for the selection of an explanation method for CNN models.

As described in the Section about the results of the experiments, the selection process computed on HPC with GPU was over 7 times faster than on PC with CPU only. Also, the Saliency method for all three problems was one of the two best methods selected. We believe, that the results of the conducted experiments can be useful for future users of the AutoeXplain package in various ways. Either to have baseline information about working with the package and the resources it needs to properly compute attributions in a reasonable time or to learn about possible results. Additionally, the results of the experiments show that each problem needs its own explanation method, even when the same model architecture is used.

**Future work** The AutoeXplain package was implemented as a part of the Bachelor's diploma thesis. Our solution uses four metrics and four explanation methods. The implementation of the package enables the future development of it for example by adding more methods to be ranked. In the Captum and the Quantus libraries, which were used for this thesis, many more methods and metrics are already implemented and can be used. Additionally, the package is currently limited to CNN models only. However, it is possible to extend it to support other types of neural networks too. Currently, parameters for all explanation methods are set during initialization and they are not modified in any way during the selection process. However, it is possible to add optimization of explanation methods' parameters for the selected method for a given problem.

# Division of work

The authors of this thesis: Agata Kaczmarek, Mateusz Stączek and Paweł Wojciechowski, worked together as presented in Table 5.1.

Table 5.1: Division of work

| Task | | Author |
|---|---|---|
| Initial planning of thesis | | Joint work |
| Planning package structure | | Joint work |
| Implementing metrics, work with HPC cluster | | Agata Kaczmarek |
| Training models, generating reports | | Mateusz Stączek |
| Implementing explanation methods | | Paweł Wojciechowski |
| Documentation | | Joint work |
| Thesis text | Chapters 1, 3, 4.2, 4.3, 5 | Agata Kaczmarek |
| Thesis text | Chapter 2, 4.1 | Mateusz Stączek |
| Thesis text | Chapter 1, 3 | Paweł Wojciechowski |
| Thesis text | Abstract, Bibliography | Joint work |


## Acknowledgments

This research was carried out with the support of the Laboratory of Bioinformatics and Computational Genomics and the High Performance Computing Center of the Faculty of Mathematics and Information Science Warsaw University of Technology.

# Bibliography


[1] M. Molina and F. Garip, "Machine Learning for Sociology", *Annual Review of Sociology*, vol. 45, pp. 27–45, 2019.

[2] K. Kourou, T. P. Exarchos, K. P. Exarchos, M. V. Karamouzis, and D. I. Fotiadis, "Machine learning applications in cancer prognosis and prediction", *Computational and Structural Biotechnology Journal*, vol. 13, pp. 8–17, 2015.

[3] W. Samek, T. Wiegand, and K.-R. Müller, "Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models", *arXiv preprint arXiv:1708.08296*, 2017.

[4] E. Tjoa and C. Guan, "A Survey on Explainable Artificial Intelligence (XAI): Towards Medical XAI", *IEEE transactions on neural networks and learning systems*, vol. 32, no. 11, pp. 4793–4813, 2020.

[5] S. M. Lundberg and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions", *Advances in neural information processing systems*, vol. 30, 2017.

[6] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic Attribution for Deep Networks", *Proceedings of the 34th International Conference on Machine Learning*, pp. 3319–3328, 2017.

[7] 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", *International Journal of Computer Vision*, vol. 128, no. 2, pp. 336–359, 2020.

[8] C.-K. Yeh, C.-Y. Hsieh, A. S. Suggala, D. I. Inouye, and P. Ravikumar, "On the (In)fidelity and Sensitivity for Explanations", *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pp. 10 967–10 978, 2019.

[9] J. van der Waa, E. Nieuwburg, A. Cremers, and M. Neerincx, "Evaluating XAI: A comparison of rule-based and example-based explanations", *Artificial Intelligence*, vol. 291, 2021.

[10] M. Nauta, J. Trienes, S. Pathak, *et al.*, "From Anecdotal Evidence to Quantitative Evaluation Methods: A Systematic Review on Evaluating Explainable AI", *arXiv preprint arXiv:2201.08164*, 2022.

[11] R. Cugny, J. Aligon, M. Chevalier, G. R. Jimenez, and O. Teste, "AutoXAI: A Framework to Automatically Select the Most Adapted XAI Solution", *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, CIKM '22, pp. 315–324, 2022.

[12] D. Alvarez-Melis and T. S. Jaakkola, "Towards Robust Interpretability with Self-Explaining Neural Networks", *Advances in Neural Information Processing Systems*, vol. 31, pp. 7775–7784, 2018.

[13] L. Rieger and L. K. Hansen, "IROF: A low resource evaluation metric for explanation methods", *Workshop AI for Affordable Healthcare*, 2020.

[14] P. Chalasani, J. Chen, A. R. Chowdhury, S. Jha, and X. Wu, "Concise Explanations of Neural Networks using Adversarial Training", *Proceedings of the 37th International Conference on Machine Learning*, 2020.

[15] H. Mueller and A. Holzinger, "Kandinsky Patterns", *Artificial Intelligence*, vol. 300, 2021.

[16] H. Tabrizchi, A. Mosavi, Z. Vamossy, and A. Varkonyi-Koczy, "Densely Connected Convolutional Networks (DenseNet) for Diagnosing Coronavirus Disease (COVID-19) from Chest X-ray Imaging", *2021 IEEE International Symposium on Medical Measurements and Applications (MeMeA)*, pp. 1–5, 2021.

[17] H. Jeremy, *Imagenette: A smaller subset of 10 easily classified classes from ImageNet*, https://github.com/fastai/imagenette/, 2019.

[18] W. Samek, G. Montavon, S. Lapuschkin, C. J. Anders, and K.-R. Müller, "Explaining Deep Neural Networks and Beyond: A Review of Methods and Applications", *Proceedings of the IEEE*, vol. 109, no. 3, pp. 247–278, 2021.

[19] M. T. Ribeiro, S. Singh, and C. Guestrin, ""Why Should I Trust You?": Explaining the Predictions of Any Classifier", *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pp. 1135–1144, 2016.

[20] M. T. Ribeiro, S. Singh, and C. Guestrin, "Anchors: High-precision model-agnostic explanations", *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018.

[21] D. Balduzzi, M. Frean, L. Leary, J. Lewis, K. W.-D. Ma, and B. McWilliams, "The Shattered Gradients Problem: If resnets are the answer, then what is the question?", *International Conference on Machine Learning*, pp. 342–350, 2017.

[22] V. Petsiuk, A. Das, and K. Saenko, "RISE: Randomized Input Sampling for Explanation of Black-box Models", *British Machine Vision Conference*, 2018.

[23] M. D. Zeiler and R. Fergus, "Visualizing and Understanding Convolutional Networks", *Computer Vision – ECCV 2014*, pp. 818–833, 2014.

[24] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", *Workshop at International Conference on Learning Representations*, 2014.

[25] L. Arras, A. Osman, and W. Samek, "CLEVR-XAI: A benchmark dataset for the ground truth evaluation of neural network explanations", *Information Fusion*, vol. 81, pp. 14–40, 2022.

[26] E. Amparore, A. Perotti, and P. Bajardi, "To trust or not to trust an explanation: Using LEAF to evaluate local linear XAI methods", *PeerJ Computer Science*, vol. 7, 2021.

[27] S. Jesus, C. Belém, V. Balayan, *et al.*, "How can I choose an explainer? An Application-grounded Evaluation of Post-hoc Explanations", *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, pp. 805–815, 2021.

[28] J. Zhou, A. H. Gandomi, F. Chen, and A. Holzinger, "Evaluating the Quality of Machine Learning Explanations: A Survey on Methods and Metrics", *Electronics*, vol. 10, no. 5, p. 593, 2021.

[29] Y.-S. Lin, W.-C. Lee, and Z. B. Celik, "What Do You See? Evaluation of Explainable Artificial Intelligence (XAI) Interpretability through Neural Backdoors", *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, pp. 1027–1035, 2021.

[30] F. Doshi-Velez and B. Kim, "Towards A Rigorous Science of Interpretable Machine Learning", *arXiv preprint arXiv:1702.08608*, 2017.

[31] W. Samek, A. Binder, G. Montavon, S. Lapuschkin, and K.-R. Müller, "Evaluating the Visualization of What a Deep Neural Network Has Learned", *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 11, pp. 2660–2673, 2017.

[32] M. Ancona, E. Ceolini, C. Öztireli, and M. Gross, "Towards better understanding of gradient-based attribution methods for Deep Neural Networks", *International Conference on Learning Representations*, 2018.

[33] A. Hedström, L. Weber, D. Krakowczyk, *et al.*, "Quantus: An Explainable AI Toolkit for Responsible Evaluation of Neural Network Explanations and Beyond", *Journal of Machine Learning Research*, vol. 24, no. 34, pp. 1–11, 2023.

[34] T. Vermeire, T. Laugel, X. Renard, D. Martens, and M. Detyniecki, "How to Choose an Explainability Method? Towards a Methodical Implementation of XAI in Practice", *Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, Communications in Computer and Information Science, pp. 521–533, 2021.

[35] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors", *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[36] A. LeNail, "NN-SVG: Publication-Ready Neural Network Architecture Schematics", *Journal of Open Source Software*, vol. 4, no. 33, p. 747, 2019.

[37] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition", *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[38] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks", *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2261–2269, 2017.

[39] C. Zhang, P. Benz, D. M. Argaw, *et al.*, "ResNet or DenseNet? Introducing Dense Shortcuts to ResNet", *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 3549–3558, 2021.

[40] V. Arya, R. K. Bellamy, P.-Y. Chen, *et al.*, "One Explanation Does Not Fit All: A Toolkit And Taxonomy Of AI Explainability Techniques", *INFORMS Annual Meeting*, 2021.

[41] W. J. Murdoch, C. Singh, K. Kumbier, R. Abbasi-Asl, and B. Yu, "Definitions, methods, and applications in interpretable machine learning", *Proceedings of the National Academy of Sciences*, vol. 116, no. 44, pp. 22 071–22 080, 2019.

[42] T. Makino, S. Jastrzębski, W. Oleszkiewicz, *et al.*, "Differences between human and machine perception in medical diagnosis", *Scientific reports*, vol. 12, no. 1, pp. 1–13, 2022.

[43] C. Molnar, "Interpretable Machine Learning, A Guide for Making Black Box Models Explainable", 2022.

[44] M. Watson, B. A. S. Hasan, and N. Al Moubayed, "Agree to Disagree: When Deep Learning Models with Identical Architectures Produce Distinct Explanations", *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 875–884, 2022.

[45] J. Adebayo, J. Gilmer, M. Muelly, I. Goodfellow, M. Hardt, and B. Kim, "Sanity Checks for Saliency Maps", *Advances in neural information processing systems*, vol. 31, 2018.

[46] P.-J. Kindermans, S. Hooker, J. Adebayo, *et al.*, "The (Un) reliability of Saliency Methods", *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, pp. 267–280, 2019.

[47] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk, "SLIC Superpixels Compared to State-of-the-Art Superpixel Methods", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 11, pp. 2274–2282, 2012.

[48] N. Hurley and S. Rickard, "Comparing Measures of Sparsity", *IEEE Transactions on Information Theory*, vol. 55, no. 10, pp. 4723–4741, 2009.

[49] A. Paszke, S. Gross, F. Massa, *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library", *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[50] N. Kokhlikyan, V. Miglani, M. Martin, *et al.*, "Captum: A unified and generic model interpretability library for PyTorch", *arXiv preprint arXiv:2009.07896*, 2020.

[51] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python", *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[52] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database", *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.

## List of symbols and abbreviations

| | |
|---|---|
| AI | Artificial Intelligence |
| CNN | Convolutional Neural Network |
| XAI | Explainable Artificial Intelligence |
| Grad-CAM | Gradient–weighted Class Activation Mapping (explanation method) |
| LIME | Local Interpretable Model-agnostic Explanations (explanation method) |
| SHAP | SHapley Additive exPlanation (explanation method) |
| IROF | Iterative Removal of Features (evaluation metric) |
| BAC | Balanced Accuracy Score |
| AOC | Area Over the Curve |
| HPC | High Performance Computing |

# List of Figures

# List of Tables

## List of appendices

# A. Generated PDF report for Chest X-ray dataset

AutoeXplainer Report

## General information

**Model name:** *DenseNet121*
**Execution time:** *11968.576 s*
**Package version:** *0.0.3*
**Date:** *2023-01-24*
**Selected method:** *Saliency*
**Number of images:** *200*

## Model performance

**Accuracy:** *0.935*
**F1 macro:** *0.935*
**Balanced accuracy:** *0.93*

## Table of results

| Explanation Name | Rank | Faithfulness Est. ↑ | Avg Sensitivity ↓ | IROF ↑ | Sparseness ↑ | Time elapsed [s] | Agg. Score |
|---|---|---|---|---|---|---|---|
| Saliency | 1 | 0.071 | 0.063 | 36.692 | 0.588 | 9.771 | 8 |
| GradCam | 2 | 0.245 | 0.142 | 42.318 | 0.377 | 4.082 | 7 |
| Integrated Gradients | 3 | -0.144 | 0.049 | 31.993 | 0.582 | 12.214 | 5 |
| KernelSHAP | 4 | 0.025 | 0.306 | 37.367 | 0.503 | 179.245 | 4 |

**Table description**
Arrow next to the metric names indicates whether larger or smaller values of metric are better. Time elapsed shows time that was required for computation of attribution for given batch of images. When there is a tie in Aggregated Score, the best metric is chosen based on computation time.

1

## Details

### Explanations:

- **KernelSHAP**: Uses the LIME framework to approximate Shapley values from game theory.(Lundberg and Su-In Lee, 2017)
  Explanation's parameters:
  `{ 'explanation_parameters': { 'baseline_function': baseline_color_black, 'baseline_function_name': 'black', 'n_samples': 50}, 'mask_parameters': {'n_segments': 50}}`

- **Integrated Gradients**: Approximates feature importances by computing gradients for model outputs for images from the straight line between the original image and the baseline black image. Later, for each feature, the integral is approximated using these gradients.(Sundararajan et al., 2017)
  Explanation's parameters:
  `{ 'explanation_parameters': { 'baseline_function': baseline_color_black, 'baseline_function_name': 'black', 'n_steps': 20}}`

- **GradCam**: For the selected layer and a target class, it computes gradients, multiplies its average by layer activations and returns only the positive part of the result. For images with more than one channel, it returns the positive part of the sum of results from all channels.(Selvaraju et al., 2016)
  Explanation's parameters:
  `{ 'explanation_parameters': { 'relu_attributions': True, 'selected_layer': 'features.denseblock4.denselayer16.conv2'}}`

- **Saliency**: Is based on computing gradients. The idea is to approximate CNN's output for a given class in the neighborhood of the image using a linear approximation and interpret the coefficients vector as an importance vector for all pixels.(Simonyan et al., 2013)
  Explanation's parameters:
  `{'explanation_parameters': {'abs': True}}`

**Metrics:**

- **Faithfulness Estimate**: Evaluates the relevance of the computed explanation by calculating the correlation between computed feature attribution and probability drops after removing features.(Alvarez-Melis et al., 2018)
  Metric's parameters:
  { 'call': {'device': 'cuda'},
  'init': { 'disable_warnings': True,
  'display_progressbar': False,
  'features_in_step': 256,
  'normalise': True,
  'perturb_baseline': 'black',
  'softmax': True}}

- **Average Sensitivity**: A metric that measures an average of how sensitive to perturbations the explanation method is. The implementation uses a Monte Carlo sampling-based approximation.(Yeh et al., 2019)
  Metric's parameters:
  { 'call': {'device': 'cuda'},
  'init': { 'disable_warnings': True,
  'display_progressbar': False,
  'lower_bound': 0.2,
  'norm_denominator': fro_norm,
  'norm_numerator': fro_norm,
  'normalise': True,
  'nr_samples': 20,
  'perturb_func': uniform_noise,
  'perturb_radius': 0.2,
  'similarity_func': difference}}

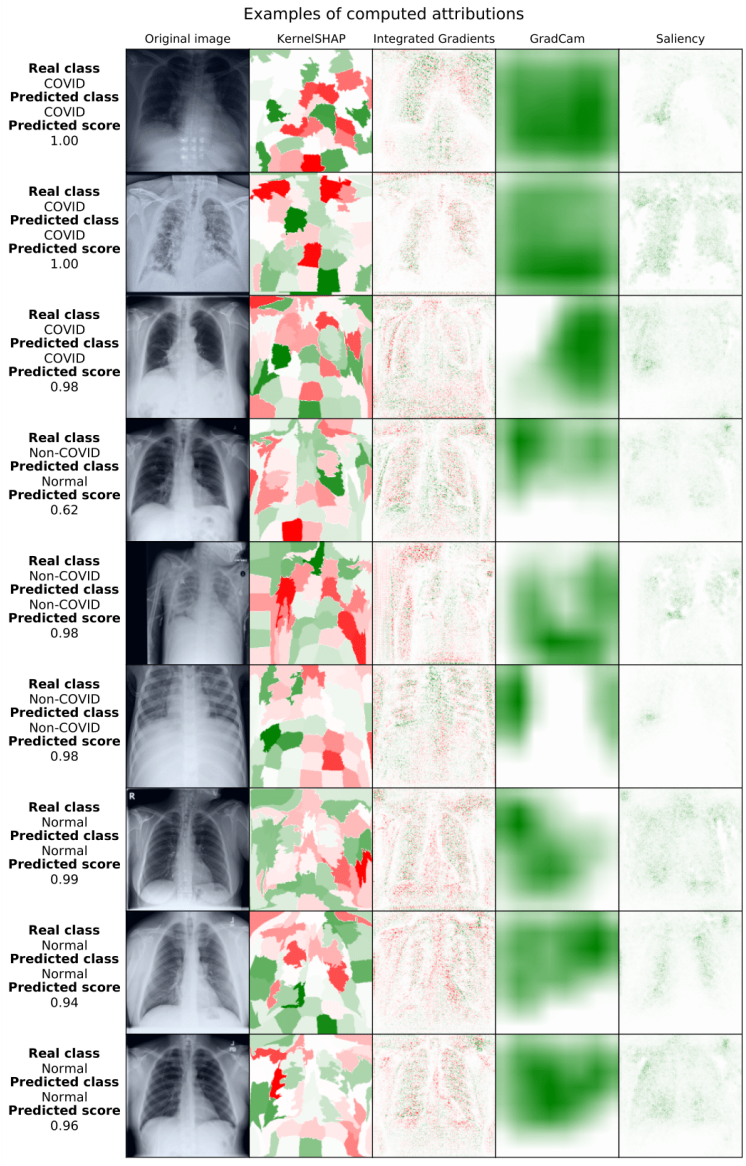
- **Iterative Removal of Features**: Iteratively removes the most important features and measures the change in probability in the model prediction for a given class. It plots the probability for a given class with respect to the number of removed features and computes the area over the curve.(Rieger at el., 2020)
  Metric's parameters:
  { 'call': {'device': 'cuda'},
  'init': { 'disable_warnings': True,
  'display_progressbar': False,
  'perturb_baseline': 'mean',
  'return_aggregate': False,
  'segmentation_method': 'slic',
  'softmax': True}}

- **Sparseness**: With the use of the Gini Index measures how imbalanced feature importances given by the explanation method are.(Chalasani et al., 2020)
  Metric's parameters:
  { 'call': {'device': 'cuda'},
  'init': {'disable_warnings': True, 'display_progressbar': False}}

**Aggregation parameters**

{ 'first_stage_aggregation_function': 'mean',
'second_stage_aggregation_function': 'rank_based',
'second_stage_aggregation_function_aggregation_parameters': {}}

## Examples of explanations




5

# B. Generated HTML report for Chest X-ray dataset

## AutoeXplainer Report

**Model name:** DenseNet121

**Date:** 2023-01-24

**Package version:** 0.0.3

**Number of images:** 200

**Execution time:** 11968.576s

**Selected method:** Saliency

### Model performance

Accuracy: 0.935 F1 macro: 0.935 BAC: 0.93

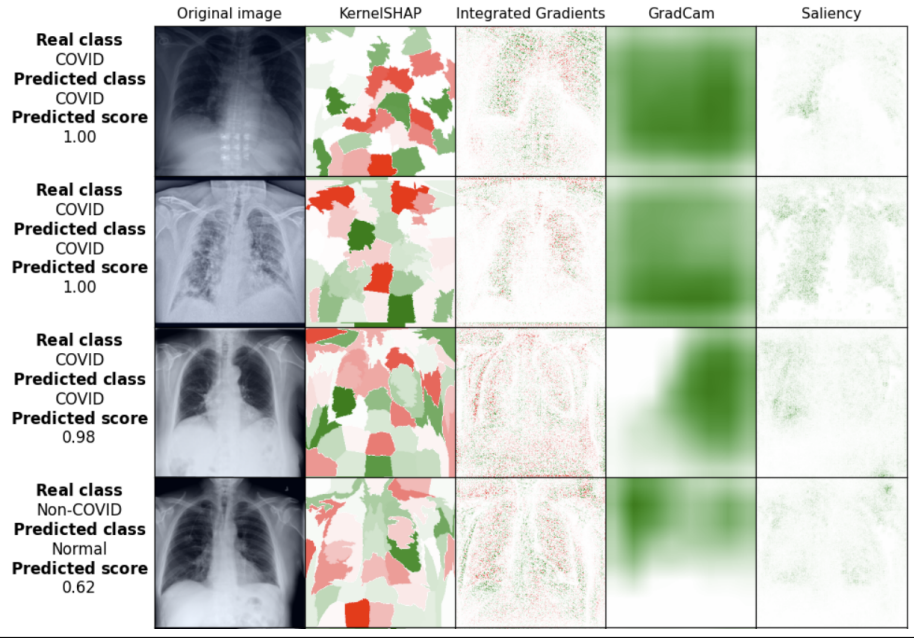
### Table of results

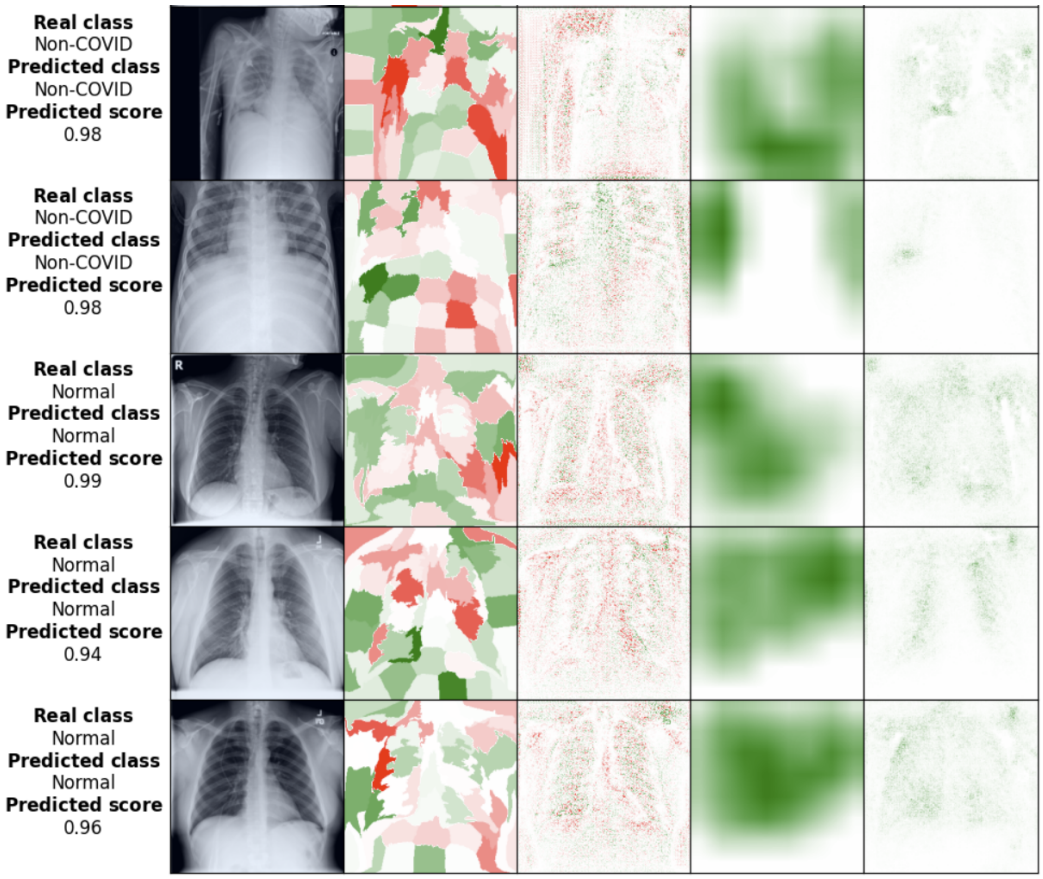
| Explanation Name | Rank | Faithfulness Est. ↑ | Avg Sensitivity ↓ | IROF ↑ | Sparseness ↑ | Time elapsed [s] | Agg. Score |
|---|---|---|---|---|---|---|---|
| **Saliency** | 1 | 0.071 | 0.063 | 36.692 | 0.588 | 9.771 | **8** |
| **GradCam** | 2 | 0.245 | 0.142 | 42.318 | 0.377 | 4.082 | **7** |
| **Integrated Gradients** | 3 | -0.144 | 0.049 | 31.993 | 0.582 | 12.214 | **5** |
| **KernelSHAP** | 4 | 0.025 | 0.306 | 37.367 | 0.503 | 179.245 | **4** |

**Table description**

Arrow next to the metric names indicates whether larger or smaller values of metric are better. Time elapsed shows time that was required for computation of attribution for given batch of images. When there is a tie in Aggregated Score, the best metric is chosen based on computation time.

Examples of computed attributions






## Aggregation parameters

```
{   'first_stage_aggregation_function': 'mean',
    'second_stage_aggregation_function': 'rank_based',
    'second_stage_aggregation_function_aggregation_parameters': {}}
```

## Explanations

**KernelSHAP:** Uses the LIME framework to approximate Shapley values from game theory. **(Lundberg and Su-In Lee, 2017)**

*Explanation's parameters:*

```
{   'explanation_parameters': {   'baseline_function': baseline_color_black,
                                  'baseline_function_name': 'black',
                                  'n_samples': 50},
    'mask_parameters': {'n_segments': 50}}
```

**Saliency:** Is based on computing gradients. The idea is to approximate CNN's output for a given class in the neighborhood of the image using a linear approximation and interpret the coefficients vector as an importance vector for all pixels. **(Simonyan et al., 2013)**

*Explanation's parameters:*

```
{'explanation_parameters': {'abs': True}}
```

**GradCam:** For the selected layer and a target class, it computes gradients, multiplies its average by layer activations and returns only the positive part of the result. For images with more than one channel, it returns the positive part of the sum of results from all channels. **(Selvaraju et al., 2016)**

*Explanation's parameters:*

```
{   'explanation_parameters': {   'relu_attributions': True,
                                  'selected_layer': 'features.denseblock4.denselayer16.conv2'}}
```

**Integrated Gradients:** Approximates feature importances by computing gradients for model outputs for images from the straight line between the original image and the baseline black image. Later, for each feature, the integral is approximated using these gradients. **(Sundararajan et al., 2017)**

*Explanation's parameters:*

```
{   'explanation_parameters': {   'baseline_function': baseline_color_black,
                                  'baseline_function_name': 'black',
                                  'n_steps': 20}}
```

## Metrics

**Faithfulness Estimate:** Evaluates the relevance of the computed explanation by calculating the correlation between computed feature attribution and probability drops after removing features. **(Alvarez-Melis et al., 2018)**

*Metric's parameters:*

```
{   'call': {'device': 'cuda'},
    'init': {   'disable_warnings': True,
                'display_progressbar': False,
                'features_in_step': 256,
                'normalise': True,
                'perturb_baseline': 'black',
                'softmax': True}}
```

**Average Sensitivity:** A metric that measures an average of how sensitive to perturbations the explanation method is. The implementation uses a Monte Carlo sampling-based approximation. **(Yeh et al., 2019)**

*Metric's parameters:*

```
{   'call': {'device': 'cuda'},
    'init': {   'disable_warnings': True,
                'display_progressbar': False,
                'lower_bound': 0.2,
                'norm_denominator': fro_norm,
                'norm_numerator': fro_norm,
                'normalise': True,
                'nr_samples': 20,
                'perturb_func': uniform_noise,
                'perturb_radius': 0.2,
                'similarity_func': difference}}
```

**Sparseness:** With the use of the Gini Index measures how imbalanced feature importances given by the explanation method are.**(Chalasani et al., 2020)**

*Metric's parameters:*

```
{   'call': {'device': 'cuda'},
    'init': {'disable_warnings': True, 'display_progressbar': False}}
```

**Iterative Removal of Features:** Iteratively removes the most important features and measures the change in probability in the model prediction for a given class. It plots the probability for a given class with respect to the number of removed features and computes the area over the curve. **(Rieger at el., 2020)**

*Metric's parameters:*

```
{   'call': {'device': 'cuda'},
    'init': {   'disable_warnings': True,
                'display_progressbar': False,
                'perturb_baseline': 'mean',
                'return_aggregate': False,
                'segmentation_method': 'slic',
                'softmax': True}}
```