



Approach to provide interpretability in machine learning models for image classification

Anja Stadlhofer¹ · Vitaliy Mezhuhev¹

Received: 10 January 2023 / Accepted: 27 June 2023
© The Author(s) 2023

Abstract

One of the main reasons why machine learning (ML) methods are not yet widely used in productive business processes is the lack of confidence in the results of an ML model. To improve the situation, interpretability methods may be used, which provide insight into the internal structure of an ML model, and criteria, based on which the model makes a certain prediction. This paper aims to consider the state of the art in interpretability methods and apply the selected methods to an industrial use case. Two methods, called LIME and SHAP, were selected from the literature and next implemented in the use case for image classification using a convolutional neural network. The research methodology consists of three parts, the first is the literature analysis, followed by the practical implementation of an ML model for image classification and the subsequent application of the interpretability methods, and the third part is a multi-criteria comparison of selected LIME and SHAP methods. This work enables companies to select the most effective interpretability method according to their use case and also to increase companies' motivation for using ML.

Keywords Machine learning · Interpretability methods · LIME · SHAP · Image classification · Convolutional neural networks · Multi-criteria analysis · Industry

1 Introduction

Industry 4.0 and Smart Factory have become common terms in a modern business environment. There are numerous projects and initiatives to push digitalization in the industrial sector. Data are the driver for Industry 4.0, and in the last years, enormous amounts of data were generated and used for analyses in many industrial fields. Companies need to respond to the digitalisation trend to stay competitive in their business areas and gain a more automatized and cost-efficient production process, leading to an economical advantage [1].

In the last few years, methods of machine learning (ML) have become increasingly important for industry digitalization. The application of ML is getting assumed as a must-have in every smart factory. ML approaches make human work less time-consuming, more efficient, and often provide

stabler results [2]. Being successfully implemented, ML models can fully automatize industrial processes, previously done by humans.

ML in image classification is a subset of artificial intelligence algorithms, which aims to make decisions based on a developed model [3]. An ML model is obtained through studying, engineering, modelling, and training with data according to the use case. An image classification model can make predictions and help in various industrial settings, for example, in digital quality control. Accordingly, the input data quality and the ML model's structure are crucial elements for image classification [2].

Due to the rising application of ML in image classification, several challenges came up. One of these challenges is the lack of transparency and interpretability of ML models and their prediction outcomes. Most of the existing models are "black boxes" that predict an outcome in a way, that humans do not understand the internal mechanisms of predictions. Understanding the behaviour of the ML model is important to give trust in the results and the ML approach itself. Therefore, explainable and interpretable ML models have become progressively important [4]. Interpretable ML models provide insights into how the model works and why it produces the

✉ Vitaliy Mezhuhev
vitaliy.mezhuhev@fh-joanneum.at
Anja Stadlhofer
anja.stadlhofer@edu.fh-joanneum.at

¹ Institute of Industrial Management, FH Joanneum University of Applied Sciences, 8605 Kapfenberg, Austria

results. There exist several methods concerning how to achieve those interpretations and how to understand them [5]. One of the most common is the Local Interpretable Model-agnostic Explanations (LIME) [6] and Shapley Additive Explanations (SHAP) [7] methods.

This research develops an approach to give interpretations of the predictions for image classification models with LIME and SHAP methods. The methodology includes both theoretical (literature analysis and multicriteria evaluation) and experimental stages. The experimental setup consisted of three parts. The first step was a collection of image data for training and validating the model. The second step contained the design and training of an ML model to get prediction results. This was performed iteratively through alternating model parameters, adapting image data, and reducing environmental influences. Then, the result was validated and verified. The last part included interpreting the final model predictions to explain why the model gave an accurate prediction and to understand the mapping between the model structure and a prediction result. Therefore, the following research questions are addressed:

What are the current approaches to support interpretability in ML models for image classification?

To answer this question, literature analysis and multicriteria comparison of the existing methods are done:

What is the most effective interpretability method for a specific use case?

To answer this question, developed ML models were explored with selected interpretability methods. Their predictions were examined to compare the methods.

This paper is structured as follows. Section 1 introduces the topic and formulates the problem statement and research questions.

Section 2 gives the machine learning fundamentals. It introduces the principles of ML, focusing on convolutional neural networks (CNNs) and image classification.

Section 3 provides a theoretical overview of interpretability in ML models, focusing on the LIME and SHAP methods.

Section 4 describes the implementation part. The methodology is discussed in detail, and the results are presented.

Section 5 is the comparison and evaluation section. The interpretability methods are compared and evaluated based on the literature research chosen metrics.

Section 6 provides a conclusion that summarizes the outcome of the research and gives an outlook on future work.

2 Machine learning fundamentals

The ML approach consists in building a model based on a given data set, to make predictions without an explicitly programmed algorithm.

ML can help with a variety of different problems and has a lot of practical applications [8].

Classification algorithms solve the task of assigning a category to an item. Examples of that are classifying documents and assigning those to the categories, such as politics, sports, weather, etc., or classifying images and assigning those to the categories, such as a cat, a dog, or a bird. In some cases, the approach goes outside of discrete classes to predict real numbers with regression algorithms (e.g., stock values) [8, 9].

2.1 Convolutional neural networks

CNNs are deep Artificial Neural Networks (ANNs) that are trained with many layers [10]. The training process belongs to the category of supervised ML methods. They are mostly used for applications in computer vision, object recognition, biological computation, and image classification [11, 12]. Explanation of CNN starts with an introduction to ANN, proceeds with explaining different layers of a CNN, and finishes using CNNs for image classification.

2.1.1 Artificial neural networks

ANNs were inspired by the real-world biological neural networks in the brain [13]. The neuron has so-called dendrites which act as the signal transmitters to the cell body. The cell body processes these signals, and the axon is responsible for sending these signals to other neurons. To make this possible, the axon terminals are connected to the dendrites of another neuron. An NN is made up of one-to-many neurons which are in communication with each other. What one neuron puts out can be used as input by another neuron [14].

An ANN can be compared to a directed graph, where the nodes are the neurons, and the edges are the links between the neurons. The neuron gets a weighted sum of the outputs of neurons as input [15]. The perceptron is the simplest ANN, because it consists of one single neuron. It takes an input vector, resulting in a weighted sum, and applies a step function to the sum for the output.

The algorithm of an ANN uses mini-batches to work through the training data sets. Each time, it goes through a batch, a so-called epoch is finished. The mini-batch starts at the input layer, where it enters the network. After that, it is passed to the first hidden layer, where the output of all neurons is calculated. This output is then passed to the next layer and this process continues until all layers are done and the output layer is reached. The next step of the algorithm is to recognize output errors, which are identified through a loss function that compares the required and the actual output. To further investigate the errors, every output connection is

measured regarding how much it contributed to the made error. Thus, for every epoch, the algorithm makes a prediction and measures its error. After this, the algorithm goes through all layers backwards to measure the contribution of each connection to the made error. In addition, based on that, it adapts the connection weights which helps the network to produce fewer errors in the future [13].

2.1.2 CNN layers

The CNNs are structured in layers. The first layer is convolutional, which focuses on detecting features. Those features can be lines, edges, colour, or other visual components. The filters detect the features, and the more filters exist in the layer; the more features can be determined. The filter is specified by a hyper-parameter that controls its width and height. Furthermore, there are weights, defined between the convolutional layer and the previous layer. Those weights can be used in another convolutional layer, which reduces the processing time. The input 3D box has the same width and height as the image itself and its depth is based on the image's colour depth. The input for the next layer is a 3D box, characterized by the hyper-parameters of the previous layer [2, 16].

The purpose of a pooling layer is to decrease the size of a 3D box. It transforms each input point, arranged in groups of an image into a single value. Those groups can be compared to the pixels in an image. Pooling layers do not have weights and they do not influence the training of the network. There exist different pooling layers, where max pooling and average pooling are the most used ones. When using max pooling, each group of input points is transformed into a single pixel with the greatest value of the group. Average pooling works the same way but with the difference that the value of the transformed pixel is the average of the group [2, 16].

Another layer type is the dense layer, which connects every neuron with the neurons of the previous layer. In a CNN, it means that every output 3D box is connected to every neuron of the dense layer and passed through an activation function. Dense layers are often used at the end of a neural network. The last dense layer usually makes the classification task. Each classification class will then be one output neuron with corresponding probabilities of the classification predictions [13, 16].

Dropout layers are necessary for preventing a neural network to be overfitted. It sets in the training phase the number of input elements to zero, which can lead to higher learning rates. During the test, validation, and production phases, the dropout layer is not used. Dropout layers are used in big networks, where model overfitting is likely [2].

A convolutional neural network is made by putting all these layers together. How many layers of which type are used and depends on the use case [17].

2.2 Image classification with CNN

Image classification is one of the applications for a CNN. It takes an image as an input and generates an output that classifies the image to a certain class. The output also gives a probability of whether the image belongs to a certain class [18].

The classification task starts with data collection. Then, the data have to be split into a training data set, a test data set and a validation data set. The next step is to build the CNN according to the use case. Next, the model is trained with the training data set. During the training, the model is evaluated constantly with the validation data set [19, 20]. The model performance and accuracy are verified with the test data set.

If the classification task is binary, a single output neuron is enough. The neuron output shows the probability of the classification. In the case of more than one label, more output neurons are needed [13].

3 Interpretability of machine learning models

ML algorithms are used as black boxes and it is often not clear how they produce specific results. At the same time, and especially in critical applications, there is a need to trust the algorithm, so that it makes the correct output [21]. Interpretability in ML is the concept of understanding the prediction of a model [22].

There exist several interpretability methods for human-friendly explanations of ML models, especially their prediction outcomes [23]. Without interpretability methods, models are often evaluated only with accuracy metrics. Explaining the prediction outcome means providing textual or visual tools to display the relationship between the prediction of the model and the involved in training data. The focus of this research lies in explaining the two methods LIME and SHAP [24].

3.1 LIME

The LIME, which is short for Local Interpretable Model-agnostic Explanation, is a model-agnostic method which can be used to provide simple interpretations to model predictions. LIME is generally used for interpretations of models, which are not highly complex [22, 25]. The method works with the surrogate model technique, which suggests using a simpler model to generate a prediction in exchange for the complex original model. For ANN, this means that the original model is transferred to an interpretable model, which tries to replicate the behaviour of the original one. This interpretable simpler model then produces the explanations for the original NN model [26].

The overall idea behind LIME as an interpretability method may be explained based on Fig. 1. The model, which needs to be interpreted, is a black box model. With LIME, the goal is to create a white-box model, which is simpler and, therefore, easier to provide the local explanations for the original complex model. First, the influencing factors in the black-box model for a single point of interest need to be understood. The single point of interest, which is going to be explained, is marked as the black cross (Fig. 1). The nonlinear background in two colours represents the decisions made about whether an instance belongs to one of the two classes. Around the point of interest, a new data set is generated to create the linear white-box model (a surrogate model). This new data set consists of sample points represented as the dots in Fig. 1.

These sample points are weighted corresponding to their distance to the point of interest. Weight is graphically displayed by their size. A local linear model which is displayed as the dotted line is then trained and used for the approximation of the original model. Instead of linear models, decision trees can also be used as a choice for the local surrogate model. Therefore, a local interpretation can be achieved, but is not globally correct [22, 27].

The LIME method is implemented as a library in Python and R for creating explanations of individual predictions. The input data format can be text, images, or tabular data. LIME can be used for multi-class predictions [23]. The method was proposed in 2016 by Ribeiro et al. [24]. LIME generates explanations for single instances by mapping input data to an interpretable representation. This could be pixels when working with images or a set of words when working with text data [26].

In more detail, LIME perturbs the input data samples and observes the change in the predictions to understand the model [22]. Here, perturbation is a method that is used to explain feature relevance. This is done by measuring alterations in the prediction outputs when features are modified. To succeed in explaining feature contribution, perturbation replacing, ignoring nonsignificant features, and learning

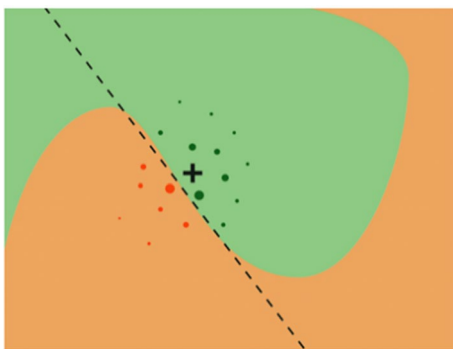


Fig. 1 Idea behind the LIME method [27]

attribution masks. LIME requires a large number of randomly perturbed samples to compute local explanations for complex ANN models. To produce an actual explanation, a prediction of the class of every sample has to be done, which needs significant computational power in the case of a large-size network [26].

3.1.1 LIME for images

Using LIME for image interpretation differs from tabular or text data. Here, the target cannot be reached by perturbing individual pixels, because a huge number of pixels is involved in predicting the class. The approach here is to segment the image into so-called superpixels that may be turned on or off. During this process, several image variations are generated. The superpixels consist of a number of interconnected pixels, having a similar colour. Superpixels can be turned on or off, where turned-off pixels are coloured in a specific colour, for example, grey [22].

The steps for creating an image interpretation with LIME are as follows. First, a prediction of the correct image class with the trained ML model is to be made. The class with the highest probability to be correctly predicted is used for the interpretation. To create an interpretation, LIME builds a new data set of random perturbations. A local surrogate model is fitted, where the computed superpixels for the input image are used to create the random perturbations. Those superpixels are computed with the quick shift algorithm. The number of perturbations is user-specific. The trained model predicts the class of each perturbed image. Before fitting the surrogate model, weights of the images, close to the original image, have to be applied to suggest the importance of the perturbed images. This importance is calculated with a distance metric that gives the gap of each perturbation to the original image (when all superpixels are on). This weighting is done for all perturbed images in the data set. Then, the surrogate model is fitted with the perturbations, predictions, and weights. As result, a factor for each superpixel is calculated, which states the effect of the superpixel on the right class prediction. The sorted factors are used to decide on the most important for the prediction superpixels. In addition, a heatmap can be overlaid to see the contribution of superpixels.

This result is the LIME interpretation, where shown superpixels have the strongest impact on the prediction. In addition, the result allows users to understand that the model makes the prediction based on significant parts of the image according to the predicted class. To validate the model, several sample images are to be interpreted, and the overall result must be evaluated [22, 28].

3.1.2 Advantages and disadvantages

Because of the representation of interpretable data, LIME is used frequently for image and text analysis. It is one of the few interpretability methods that could work with tabular data, text data and images. The inner algorithm of LIME is based on the generation of a simpler model to approximate the original model. It allows the use of the same local, interpretable model to do the interpretation [27, 29].

For an end user, it is easy to find an explanation in the interpreted parts of an image or text and to understand it. LIME is a good option for not very experienced ML users, because LIME produces human-friendly explanations.

At the same time, LIME might not be the best option to use when detailed prediction explanations are required.

For tabular data, LIME may also not be the best option, since there are issues in finding the correct interpretable representation. At the same time, there have been attempts to solve this problem with several different transformation methods.

What should also be considered when working with LIME is that the needed data points are sparse and defining a local neighbourhood is not a simple task. If the neighbourhood is changed, the explanation results are different, causing instability in the method. Due to the unstable outcomes in the repeating sampling process, there is a need to check for every iteration if interpretations make sense. As a result, LIME is a promising method for high-dimensional data, and its issues are to be overcome in its further development phases [27, 29].

3.2 SHAP

SHAP is the abbreviation for Shapley Additive exPlanations. It is an interpretability method with the idea to explain ML models using a game theory approach [7]. This approach considers a feature for a data instance as a player and the prediction as a payout. The SHAP method works based on Shapley values, which are used to rank the ML model's features. Overall, the SHAP approach considers all predictions, which makes it more reliable and ensures stable results as opposed to LIME. At the same time, this makes it more computationally time-consuming. SHAP can be used both for model-agnostic and model-specific techniques [23, 25]. It is a post-hoc method used mainly in local interpretations, but global ones are also possible. Before discussing, the SHAP values are considered in more detail.

3.2.1 Shapley values

Shapley's values came from the game theory. The relationship between them can be seen as the players being included

in the model and the game reproduces the result of the ML model [22]. For example, the Shapely values may be used to predict the price of an apartment. For example, we are looking for an apartment with the characteristics of being near a park, having 50 m² on the second floor with no cats allowed and the price of €300,000. This prediction may be explained by looking at each of these feature values individually and explaining their contribution to the overall prediction. Say, the amount of €310,000 is the average prediction the model made for all apartments. The goal now is to compare the average prediction to the prediction with emphasis on how much each feature value contributed to that. For linear models, the effect is calculated through the weight of the feature times the feature value.

SHAP estimates the effect through local models and Shapley values, by assigning payouts to players, which depends on their contribution to the total pay-out. The game in this setting is the prediction task for a single element in the data set. The earning is the made prediction minus the average prediction for all elements. The players, as mentioned before, are the feature values of the element, which help to get the earnings. In the apartment example, the prediction of €300,000 was achieved through the feature values describing its characteristics. The goal of explaining the difference was achieved by subtracting the €300,000 from the average predicted €310,000 resulting in a difference of –€10,000. The interpretation of the ML model could be, for example, that the park contributed €30,000, the area €10,000, the 2nd-floor €0 and the banned cat –€50,000 which results in the sum of –€10,000.

The Shapley value for one feature is calculated by the average marginal contribution of this feature value above all potential coalitions. A sample contribution of the banned cat feature value appended to a coalition of a park is nearby and the area can look like the following. A random apartment from the data set is selected and its value for the floor feature with a coalition of a park nearby, the banned cat and the area feature. The previously 2nd-floor feature is now set randomly as a 1st-floor feature. With this information, the price of €310,000 is predicted. The next step is to eliminate the banned cat feature and substitute it with a random allowed or banned cat feature. Then, the price with the park nearby and area coalition may be €320,000. The banned cat feature contributed –€10,000 to the prediction, calculated as €310,000 minus €320,000. These steps to create the interpretation are based on the values for the floor and cat features values from the randomly selected apartment. The estimation will improve with every sampling step, whereby the contributions are getting averaged. Such calculations need to be done for all potential coalitions resulting in the Shapley value, which is the average of all contributions to all potential coalitions [29].

Shapley values are often used in the approximate solution, because their calculating takes up a lot of time due to many possible coalitions. Furthermore, Shapley values tend to be misinterpreted in a way that they are the difference between the predicted value after deleting a certain feature from the model. Instead, a Shapley value is the contribution of a feature value to the difference that exists between the mean and the actual prediction. This interpretability method uses all features as opposed to LIME, where the option to select is given. Shapley values deliver a full explanation, provided through the full distribution among all included feature values of the data instance. In addition, comparison against a subset or a single data point is possible, not only for average predictions as LIME does [26, 29].

3.2.2 SHAP method

SHAP is implemented as a python library. Due to the assumption that the local surrogate model must not be linear, SHAP is more time-consuming than LIME [30].

SHAP provides an interpretation, based on explaining individual predictions. This is done by calculating the contribution of every feature to the prediction with the Shapley values. The feature values of an example data instance can be seen as a player in a coalition according to the game theory approach. A player can be a cluster of feature values or a single feature value. As an example, the single feature value can be distributed on superpixels for images. SHAP has three required properties. The first one is local accuracy to make the method efficient. The second one is missingness, which helps with keeping the local accuracy in place and is mostly relevant for constant features. The third property is consistency. It ensures that if the model changes by its contribution of a feature value, then the Shapley value also changes accordingly [29].

There exist several approximation methods for calculating Shapley values, where KernelSHAP and TreeSHAP [26] are most used once. KernelSHAP is a kernel-based approximation approach and TreeSHAP is efficient for tree-based models [29]. KernelSHAP is a combination of linear LIME, and Shapley values and is a model-agnostic implementation of SHAP. It is an algorithm that makes the Shapley value approximation locally referred to a data instance, by creating samples of possible coalitions. The kernel in this case has the function of weighting the coalitions [26, 31]. To be more specific, the first step to calculate the contribution of a feature to the prediction is to define sample coalitions, and then to get a prediction for each one of them. Afterwards, the SHAP kernel applies weights, and the weighted linear model is fitted to return the Shapley values to be further processed. The difference to LIME is that SHAP applies weights to the sample

instance based on the weight of the coalition and LIME applies weights based on the original instance. The idea here is to learn about features when they are isolated. For example, when the coalition has only one feature, then the effect this feature has on the prediction can be derived. The same principle applies when the coalition consists of many features. TreeSHAP uses the model-specific approach and is faster than KernelSHAP. The principle of TreeSHAP is to generate computations down the tree at the same time. TreeSHAP may have the issue that a feature, which does not influence the prediction, can be assigned other than the zero value. This is the case, when a feature correlates with another feature, significantly influencing the prediction [22].

3.2.3 SHAP for images

SHAP for images is used mostly for image classification. The idea here is to determine for every pixel in the predicted image the level of pixel contribution to a certain class. To make the interpretation, an NN model for image classification is to be trained and next used for a prediction on a test set image for every class. Finally, the SHAP values with the help of the SHAP library are generated and visualized. The interpretation of visualization of the images shows highlighted parts in shades of red and blue. The image labels show the predicted classes in descending order of probability. The red pixels indicate SHAP values that contributed in a positive way to the classification of the labelled class. The blue pixels indicate the opposite meaning, i.e., they contribute negatively to the prediction [32].

3.2.4 Conclusion

The SHAP interpretability method has a sound theoretical foundation. The method uses the distributed feature to generate the Shapley values. SHAP relates to LIME, taking the interpretable ML area a step further to be unified in an overall approach. The possibility of providing global and local explanations is also an advantage of SHAP. The TreeSHAP allows a user to create fast interpretations. The slower KernelSHAP may be applied for the models, where TreeSHAP is not feasible. KernelSHAP can be time-consuming for the computation of Shapley values if a lot of instances are involved. In addition, creating misleading interpretations is possible, for example, to hide biases [27, 29]. The final comparison and evaluation of the LIME and SHAP methods will be done in Sect. 5. The metrics will be derived from the literature research.

4 Implementation

For the implementation, a use case in the Smart Production Lab of the FH JOANNEUM University of applied sciences, Austria, was developed. The focus was on creating an ML model for image classification for quality control and on providing interpretability options for the developed model. The product of the selected use case was a watch, manufactured in the Smart Production Lab. One part of the product, namely, the watch stand, was chosen for quality control. The developed ML model classified two types of images. The image taken of the watch stand could be considered as “good”, which means that the product has the desired quality, so that it can be further processed to be assembled into the whole product. The other type of image classifies the products with defects, with the consequence that these products cannot be further processed. Therefore, the developed model distinguished between these two types, which resulted in a binary classification problem. To analyse, why the developed ML model made a certain prediction, the methods LIME and SHAP were implemented. The results were compared to provide a selection approach, helping the industry in choosing a proper method based on the use case.

4.1 Machine learning model implementation

Figure 2 shows the methodology flowchart for the implementation stage of the ML model for image classification. It is segmented into four major phases, namely, collection, preprocessing, learning, evaluation, and prediction.

The starting point was to collect data by taking product pictures, changing the lighting conditions, and adjusting angles and perspectives. This process was done with products of the class “good” and “defect”. The second phase

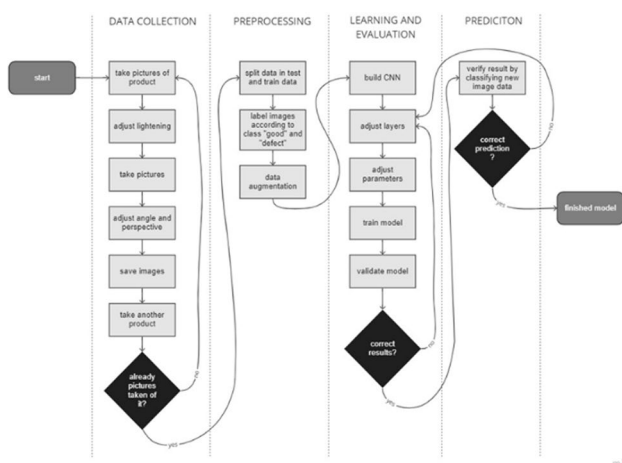


Fig. 2 Methodology flowchart of the model implementation

started with splitting the image data into a train and test set and labelling the images according to their class. The last step was data augmentation to prepare for building the CNN model. Layers and parameters were adjusted before the training starts and the trained model could be validated. If the results were not acceptable, then the adjusting repeated again. After this, the model was verified by classifying a test image data set to determine whether the model made the correct prediction. The individual steps and milestones are described next in more detail.

4.1.1 Experimental setup

The development of the ML model was done with a virtual machine on a High-Performance Cluster (HPC) provided by the FH Joanneum. The HPC is a network of servers that works in parallel to increase computing power and, therefore, makes model training faster. To upload the image data on the HPC, the open-source server- and client-software FileZilla was used. Furthermore, Anaconda was applied as the environment management system for installing, running and updating packages and their dependencies. Conda was used in combination with the programming language Python. As a development environment, the Jupyter-Notebook was applied, which is an open-source, browser-based tool that enables users to create documents with live-code, text and visualizations. For the CNN implementation, the open-source platform TensorFlow was used. TensorFlow incorporates a lot of tools and libraries for ML applications. In detail, the module Keras, which runs on top of TensorFlow, was used. It is a deep learning Application Programming Interface, written in Python for simple, flexible and powerful use to solve ML tasks. Keras has a lot of libraries, which can be used for the implementation of the ML model, especially for image classification.

4.1.2 Data collection and preprocessing

For this use case, the images of the watch stands were made with the camera of a smartphone. The camera of the brand Sony has 64 megapixels, which was enough for the distinction between the classes. Furthermore, the images were downsized to accelerate the computations. A possible series of “good” products are shown in Fig. 3. Examples of defective parts are shown in Fig. 4.

These images were organized in a directory structure for the model training and its further steps. The main data directory had two subdirectories, called “train” and “test”. In addition, these two directories both had two subdirectories called “defect” and “good”. This structure is due to using Keras, because it recognizes the classes according to the directory names. 80% of acquired data were used for model training and 20% for testing purposes. The number of images



Fig. 3 Series of product images of the class “good”

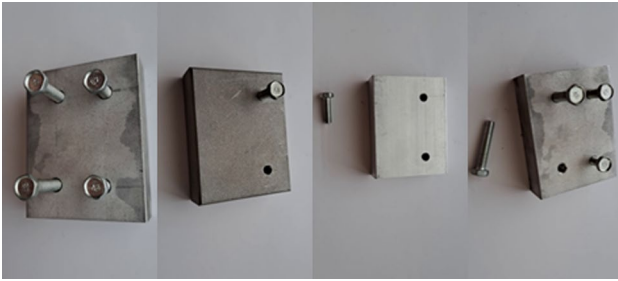


Fig. 4 Series of product images of the class “defect”

taken for one class was 6007. This number was split into the “test” and “train” directories, resulting in 4806 pictures (both “defect” and “good”) in the “train” directory and 1201 images in the “test” directory (“defect” and “good”). In total, 12,014 images were taken to generate the data to create an image classification ML model. To speed up the process, image retrieval was performed by making a video of the product parts in different angles and positions with different lighting conditions. Then, the video frames were cut out to be used as the images.

4.1.3 Model description

The implementation of the ML model was based on Keras libraries. The model consists of the core data structure of Keras, namely, layers. The simple sequential model was used, which is a linear stack of several layers. The model consisted of three major convolutional layers, three Max-Pooling layers, one flattens layer and one dense layer. This structure had been chosen by an experimental approach. This means, during the development phase, different design structures were created and evaluated. Once the desired result with the intended accuracy was achieved, this model structure was selected to work with.

The next step was compiling the model to initiate the learning process. Then, the actual training could start. This was done by iterating over the training data with the fit() function. The training was done for 10 epochs, which means that the entire training data set was passed through

Table 1 Confusion matrix

	Predicted good	Predicted defective
Actually good	1197	4
Actually defected	2	1199

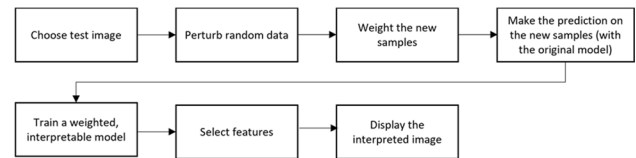


Fig. 5 Methodology chart of the LIME method

the developed CNN ten times. For every epoch, the weights were changed to create higher accuracy outputs. To optimize passing the entire data set through the network, data were divided into batches with the size 50 (the number of images used at once to pass the CNN). Iterations of batches were used until all images from the data set were through to complete one epoch. For this configuration, the training process took up around 2 min and 20 s to complete. The entire process, from loading the images to training the model to verify the results, took up 6 min and 30 s.

The overall model accuracy at the end of the training with 10 epochs was about 99%. The corresponding confusion matrix for the model validation can be seen in Table 1. The model made the prediction for every 1201 «good» and 1201 «defect» images. The model predicted 1197 images, that were labelled as good, to actually be good, and only four of these good images got predicted as defective ones. The second line in the confusion matrix says that 1199 images that are labelled as «defect» were classified correctly and only two of those were classified as «good» ones.

4.2 LIME interpretation methodology

The methodology of creating the interpretation with LIME included several steps (Fig. 5). The first step is to choose an image from the test data set. After this, the data were randomly perturbed by turning superpixels on and off and those newly generated samples were weighted. The weighting of the samples was done by the means of their proximity to the region of interest. Based on the new samples, a prediction was made with the original model. Thereafter, a new weighted model with the newly generated data set was trained and referred to as the interpretable or the local surrogate model. After the feature selection, the local surrogate model was interpreted with the prediction on the test image and its result was displayed.

The idea behind the process is that LIME uses decision boundaries for the two classes. The prediction of an instance, also called a data point, gets then the explanation. This explanation is generated by creating a new data set of around the data point's located perturbations. Every perturbation gets predicted from the ML model and classified into the class “good” or “defect”. Every perturbation has a level of importance. Perturbations are more important when the distance to the original data point is small. Next, these distances are used to calculate the weights. These preparations are used to create the local surrogate model. The key components to make the interpretation, therefore, are the newly generated data set, the predictions on the new samples and their weights.

4.2.1 Software implementation

For implementation, the LIME libraries of version 0.2.0.1 were used. The LimeImageExplainer was used to create the LIME explainer instance. The parameters were the image, on which the interpretation should be done and the model to be used for the prediction. The top_labels parameter induces to have a look at the top two predictions for the test image, since in this use case there were only two classes. The hide_color parameter represents the colour for a superpixel that is turned off (i.e., it is not relevant for the prediction made). In addition, the num_samples parameter sets the number of the generated artificial data points.

Applying the LIME explainer results in the explanation of the prediction in the form of the LIME interpretation. The first image in Fig. 6 shows the top five superpixels that are the most important towards the correctly classified class. The rest of the image is hidden. The second image in Fig. 6 shows the same interpretation but in a different representation style.

The red highlighted parts equal the hidden parts of the first image and indicate that these parts contribute negatively or have no impact on the correctly made prediction. Whereas the green highlighted parts contribute positively to the made prediction and played a significant role in it. The coloured

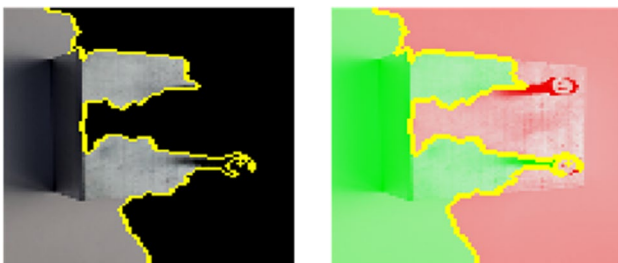


Fig. 6 LIME interpretation image

parts show the increase or decrease in the probability of the image to be classified to the first or the second class.

Another explanation method of LIME interpretation is to use a heatmap, which plots the explanation weights (see Fig. 7). The colorbar on the right displays the values of the weights in the image. To create this heatmap, every explanation weight was mapped to its corresponding superpixel and then plotted using different colours. With that, a more detailed insight into the interpretation can be given, showing how the superpixels are contributed. For example, the light pink part in the middle is an important one, belonging to the positively contributed part.

4.2.2 Result interpretation

Figures 8 and 9 allow us to get a better understanding of the LIME interpretations on different sample images. In these images, the green highlighted parts represent the positively contributed superpixels to the correct prediction and the red highlighted parts are the negatively or not contributed superpixels. For all presented in Figs. 8 and 9 sample images, the ML model made the correct prediction. It classified all the images in Fig. 8 as “good” products and all images in Fig. 9 as “defect” ones. With this graphically displayed interpretation, it gets clear that the edges and screws on the images of the class “good” were important for the made predictions. For the “defect” class, LIME interpretations focused on the

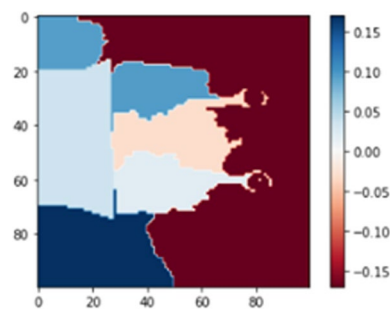


Fig. 7 Heatmap of the LIME interpretation image

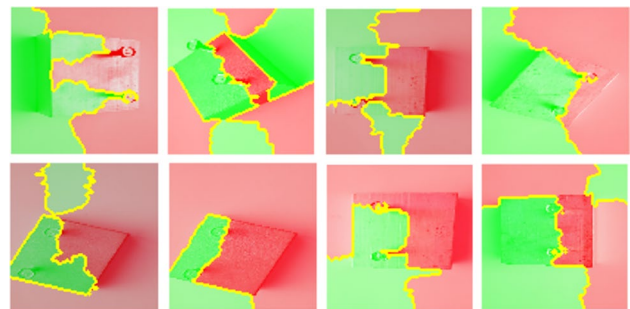


Fig. 8 Example of LIME interpretations of the class “good”

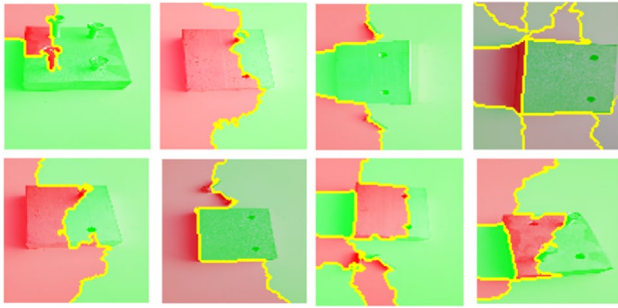


Fig. 9 Example of LIME interpretations of the class “defect”

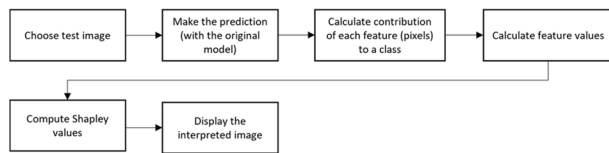


Fig. 10 Methodology process chart of the SHAP method

edges and the screws of the product; and also, the holes played a significant role. It coincides with the way a human would classify the images, which makes the model more comprehensible and promotes confidence towards its real-life application in the industry.

To describe the interpretation in more detail, the first image of Fig. 8 may be used. In this image, the bottom screw is the most important for the correct prediction. The parts of the image towards the upper screw increased the probability the image was classified into the correct class “good”. In the first image of Fig. 9, it appeared that nearly part of it contributed positively to the correct prediction. However, one corner with an exactly framed screw is highlighted red. This means that the model considered this screw as a part of a “good” product, but because of the other three screws, the model classified it to the class “defect”. In this way, a close look at interpreted images is needed to understand the LIME interpretation and to encourage the model’s decisions. Another result can be that the model makes its prediction based on irrelevant parts of the image and thus it should not be trusted.

4.3 SHAP implementation

Figure 10 summarizes the most important steps of a SHAP interpretation. SHAP is also a post-hoc method, meaning the interpretation is done on a single image from the test data set after the model training. The original ML model makes the prediction on the chosen test image. Then, the contribution of each feature is calculated. The obtained information is used to compute the Shapley values to make the interpretation. The last step is to display the interpretation by plotting

the test image with the visualized SHAP interpretation over it.

SHAP interpretation process aims at explanation of the prediction of a single instance in the test image. This is achieved by calculating the contribution of each feature to a certain prediction class. Shapley values are computed to provide information on distribution of the prediction among the features. For image classification, the pixels in an image are grouped into superpixels and the prediction is allocated to them.

4.4 Software implementation

Programming of the SHAP interpretation method starts with importing and initializing the shap Python package. In the implementation, SHAP of the version 0.40.0 was used. The two class labels “good” and “defect” were defined. After this, an image from the test data set was chosen and loaded into the Jupyter-Notebook. To create the visual explanation of the test image, a mask to be defined for blurring the interpretation of the original image. The shape of the image needed to be defined to place the correct fitting mask over the original image. The next step is to use the SHAP library to generate the SHAP values. The primary explainer of SHAP was used to create the interpretation of the prediction of the image. This explainer using every combination of the model and the masker to return a subclass object. This is achieved with the explainer constructor with parameters: the trained CNN model, the previously described masker and the class labels in the form of a list. The parameters of the object include the image on which the interpretation is to be made and the number of sample images that should be taken from the test data set. Furthermore, the number of evaluations of the ML to estimate the SHAP values were defined. This number defines the duration of the explanation, and correspondingly, the quality of the approximation. In this use case, the size of the batch of the evaluation was set to 50. The output includes two interpreted images and the original image. The last step is then to plot the SHAP values to see the interpretation visually.

Applying the SHAP values and plotting the result can be seen in Fig. 11. The first image is the original image from the test data set. The second image shows the SHAP interpretation and provides the information of the prediction result. Here, the test image was classified as “good”. The third image is assigned to the class with the second highest probability. Note, it is possible to show more than two classes for multiclass classification problems. Whether the prediction was done correctly or not can be seen in the plotted SHAP result. The image highlights the SHAP interpretation in the shades of red and blue (in the background the original product can be seen).

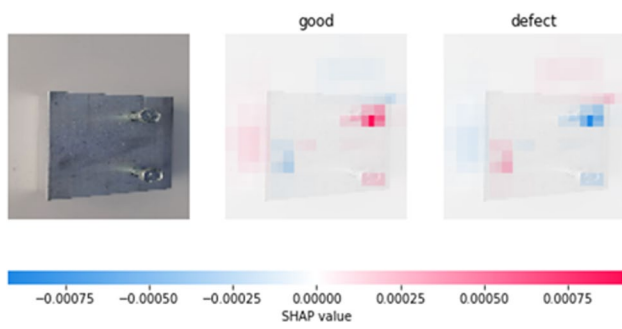


Fig. 11 SHAP interpretation image

The highlighted by red parts mean that they contributed positively to the made prediction. In addition, the blue highlighted parts contributed negatively to the prediction. Working with superpixels makes it easier to recognize the parts in the image that are important for the prediction. The darker the shade of red or blue is the higher is the contribution to the classification of the particular class. This means the parts, which were shaded darker are more important vs the pale colors. It can also be seen by the colorbar at the bottom, which represents the SHAP values corresponding to the color shade.

4.4.1 Result interpretation

The SHAP method was executed on different sample images from the test data set. A collection of the class “good” is shown in Fig. 12 and of the class “defect” in Fig. 13. Both

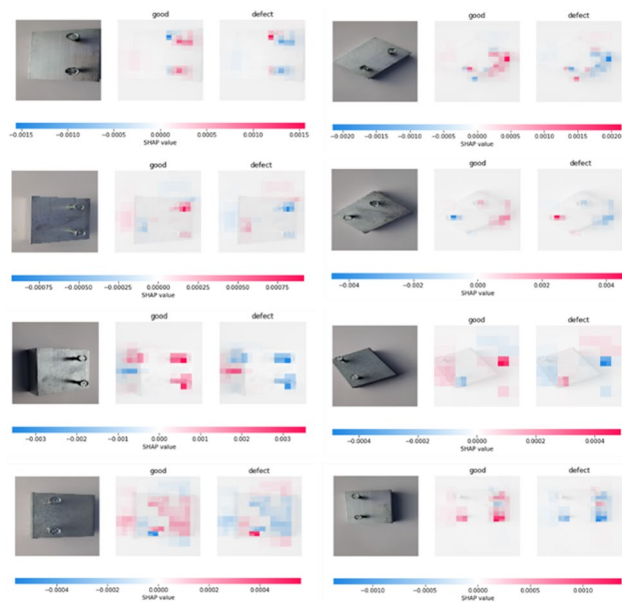


Fig. 12 Example collection of SHAP interpretations of the class “good”

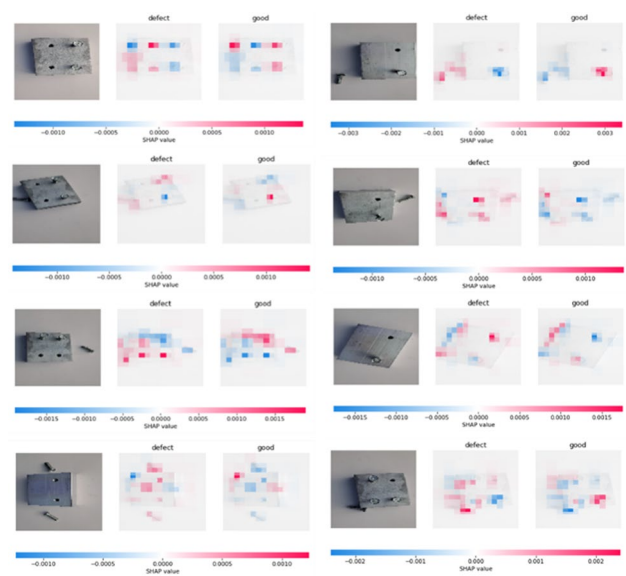


Fig. 13 Example collection of SHAP interpretations of the class “defect”

collections contain eight randomly chosen images from the test data set that got interpreted with the SHAP method. On these images, the red highlighted parts represent the positively contributed superpixels to the correct prediction and the image parts with the blue highlighted superpixels contributed negatively to the prediction. All these 16 example images were classified correctly to their classes by ML model. Unlike the visual interpretation of LIME, the edges of the products in case of the SHAP do not play an important role in the explanation of the prediction. At the same time, important become the screws, which are placed correctly in the “good” images, misplaced besides the product and also holes without screws in the “defect” images.

To describe the interpretation of SHAP in more detail, let us consider the first plotted result in Fig. 12. It shows the product of the class “good” with two correctly placed screws. The visualized interpretation presented besides the actual image highlighted superpixels in shades of red and blue according to the computed SHAP values. The darkest highlighted parts are shown in the places of the screws. This makes it easy to understand why the model classified this image as “good”. There are also red highlighted parts but with a lighter shade. For this and for other images, the black shadow of the screw is sometimes considered as a hole, which is a significant feature for a “defect” product. These are then highlighted blue, because the ML model would classify a product with a hole as “defect”, even though the product is “good”. Thus, these pixels contributed negatively to the result. However, the image is still classified as “good”, because the other features are superior. It can be seen for the second classification with the label “defect”

that the highlighted pixels have exactly the opposite colour on the same place. This shows that if the image had been classified as “defect”, then the parts of the image that are actually considered “good” in that case contribute negatively to the prediction.

Let us also consider the first image of the class “defect” in Fig. 13. This example has nearly the same amount of highlighted red and blue parts, which means that it was not completely sure for the ML model to which class the image belongs. The two holes in the product contributed positively to the correctly made prediction, since they were highlighted in a darker-shaded red. However, the two additional screws were highlighted blue, which means that they contributed in a negative way. This behaviour can be explained by the fact that “good” products have exactly two screws in the same position. These are the features that contributed to the “good” classification. Another aspect here is the shadow of the product, which showed these parts of the image as important for the prediction. Looking at sample images which were interpreted with SHAP, it gets clear based on which features the ML model makes its classification prediction. Having this knowledge, the ML model can be improved further, for example, by eliminating shadows that interfere with the significant parts of the image.

5 Comparison and evaluation

This section compares LIME and SHAP methods through functionality metrics, derived from the literature.

5.1 Computational time to explain the prediction

The computational times, needed to derive the explanation of the prediction, were retrieved during the execution of the Python code for each method in the same controlled conditions. There were two types of times measured. The first one was the CPU time (execution time). It measures how much time has elapsed until the CPU has executed the core program (without initializing variables and plotting the result). The second one was the wall time (running time), which measures the total time to execute the program. Since the execution of the program was done on a high-performance cluster, the wall time was smaller than the CPU time to compute the interpretation. For LIME, the CPU time for the same ML model was 12.2 s, and for SHAP, it was 11.8 s. The wall time for LIME was 4.8 s and for SHAP 2.29 s. Thus, for this specific use case, program implementation of the SHAP method is slightly faster than LIME. In addition, based on the literature, LIME is generally faster when not used for image data, but for text or tabular data.

5.2 Presentation of the interpretation

Both methods present their results visually in a simple, and easy-to-understand form. The representation of the explained prediction in both cases is an overlay to the original image from a test data set. Indicating the positive or negative contributed parts is made by the colour palette, both for LIME and SHAP. However, LIME uses only two colours, which highlights either a strictly positive or negative contribution. Whereas SHAP uses different shades of the two colours, which makes the interpretation more expressive. Thus, SHAP provides more information, and a conclusion can be drawn, which positively contributed parts are more important than others. A similar option provides the LIME heatmap, by segmenting the superpixels according to their weights, but this is only to be seen as additional information. To conclude on the explanatory power, SHAP appears to have a more informative way of expressing and representing the interpretation of the ML model prediction.

5.3 Interpretability

LIME and SHAP are primarily local interpretability methods, meaning that they are unaware of the inner structure of the model. The option that SHAP provides for global interpretability is to sum all individual predictions of the SHAP values. Although both methods are primarily local, they use different approaches. LIME builds local surrogate linear models for each prediction that gets explained. This creates a white-box model from the initial black-box model. However, this approach is limited to the local neighbourhood of the model. SHAP uses the Shapley values to determine the average marginal contribution of all feature values for all possible coalitions. This means that SHAP investigates all possible predictions of the image or non-image data. This approach ensures that the interpretations of SHAP are accurate and consistent. In some literature sources, it is suggested that LIME is a subset of SHAP with a lack of consistency or accuracy.

5.4 Applicability

LIME and SHAP are the most common methods in ML interpretability. Therefore, they are applicable to different use cases and different data types in industrial settings. Both methods have Python implementation, which is currently the most used programming language in ML applications. We need to note that the selection of the method may depend on a specific ML algorithm. In this use case, both methods were applied for CNN. At the same time, for a model built with the k-nearest neighbour algorithm, computing the SHAP values will take a long time, in comparison with LIME. Furthermore, using LIME and

SHAP on Keras machine learning models works out of the box. However, LIME, for example, cannot be used on an XGBoost machine learning model without creating a workaround. In conclusion, LIME and SHAP are generally well-applicable in industrial settings in contrast to similar methods, for example, Anchors.

5.5 Replicability and reliability

Recomputing the SHAP values on similar images will always result in a similar explanation output. Thus, SHAP is a stable interpretability method, where the explanation output can be replicated. On the other hand, for the LIME, explanation outputs for similar images can be different. This problem arises because of the rather weak approximated local surrogate model in relation to the original black-box model. This is the case because of the perturbation step of LIME, which can differ when repeated. For our use case, LIME results differ slightly in the form of smaller boundary shifts from the positively or negatively contributed side. Superpixels, which were at the tipping point between the positive or negative contribution, changed in the repeated sampling process. However, the differences in the results for LIME interpretations were minimal.

5.6 Implementability effort

Both methods are easy to implement with a corresponding python library. The programming effort depends on the industrial use case, available data and the selected ML model. After the data preparation step, the right design is to be found for the ML model. In conclusion, the implementability effort of LIME or SHAP methods for a programmer is quite similar.

5.7 Limitations

Implementation of the method should be always examined, and the results should not be blindly trusted. Due to sample variations, LIME lacks a guarantee of producing stable and consistent results for similar images. Furthermore, LIME limits itself to producing local surrogate models with different quality levels, since the fit of the data to the model cannot be controlled. There are no clear instructions on how many features to select for the local surrogate LIME model, which may result in either too complex or too simple interpretation. In addition, SHAP is generally known to be slower than LIME when many instances need to be computed. This can be a significant criterion for the selection of the corresponding method in the industrial context.

6 Conclusion and future work

The development in artificial intelligence and especially ML methods is fast-moving and makes enormous progress every year [33–37]. While in some industrial areas, ML concepts are used productively, in others implementation is lagging. Industries, which want to stay competitive must deal with the introduction of ML in their corporate processes, for example, in automated quality control. At the same time, ML is often used without proper interpretation of the models. The question of why the model makes a certain prediction cannot be answered. Therefore, the use of explainability and especially, interpretability methods, becomes an increasingly important approach, which should be established in every company that uses machine learning applications.

This work introduces the ML interpretability methods for image classification in industrial use. Specifically, it implements and examines LIME and SHAP methods. To limit the area of application, the focus was given to applying the methods for a binary image classification, which was performed using a CNN algorithm. Due to a limited scope, the research insights and recommendations are valid for the developed use case. At the same time, the proposed recommendations and evaluation scheme can be used in similar use cases in the industry.

Application of the interpretability methods to the predictions of an ML model provides a better understanding of the model work. In our use case, based on the interpretability methods, the ML model for image classification was redesigned and reached an accuracy score of 99%.

Another result is giving users insights to understand the incorrect image classification, for example, that the ML model may consider a shadow of a screw as a hole (which is a criterion for a “defect” product). In general, any shadows of the product played a significant role in ML prediction. From that, it can be concluded that the experimental setup should be improved by adding light sources to minimize the shadows.

The literature research provides the answer to the first research question and selects the approaches (the LIME and the SHAP methods). The results of the comparison of the methods showed that they produce similar results based on different approaches. The SHAP method is, in some aspects, superior to the LIME method. SHAP has a more elaborated theoretical foundation behind the computation of the interpretation values and produces more stable and consistent results. In addition, in the SHAP method, the visual interpretation is more detailed and reasonable, which leads to the possibility to draw rich conclusions. While LIME was described in the literature sources as the faster alternative for SHAP, for the selected use

case, SHAP was faster than LIME. Both methods operate at a local level, which means they interpret not the whole model, but only the final predictions. Based on these results, the second research question is answered with the selection of the method SHAP. It can be recommended as the most effective option when providing interpretability for an ML model for image classification in the industrial context.

Future work would be to evaluate the selected methods on different use cases and ML models. Further research must be done, since the existing interpretability methods are only at the beginning of their development and need to be evaluated in their industrial applications towards gaining an understanding of ML results.

Author contributions We acknowledge that the authors have contributed significantly and are in agreement with the content of the manuscript.

Funding Not applicable.

Availability of data and materials Data are available upon request to the authors.

Declarations

Competing interests We acknowledge that authors have no competing interests.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Oks SJ, Fritzsche A, Lehmann C (2016) The digitalization of industry from a strategic perspective. In: Presented at the R&D management conference from science to society: innovation and value creation, Cambridge, United Kingdom
- Bonaccorso G (2017) A gentle introduction to machine learning. In: Machine learning algorithms—a reference guide to popular algorithms for data science and machine learning, Birmingham, United Kingdom, pp 6–9
- Zhang X (2020) Machine learning. A matrix algebra approach to artificial intelligence, 1st edn. Springer, Singapore, pp 223–224
- Dosilovic FK, Brcic M, Hlupic N (2018) Explainable artificial intelligence: a survey. In: Presented at the 41st international convention on MIPRO, Opatija, Croatia, pp 210–215
- Bhatt U et al (2019) Explainable machine learning in deployment. In: Presented at proceedings of the 2020 conference on fairness, accountability and transparency, Cambridge, United Kingdom
- Ribeiro MTC (2021) Lime. <https://github.com/marcotcr/lime>. Accessed 2 Jan 2022
- Lundberg S (2018) Shap documentation. <https://shap.readthedocs.io/en/latest/index.html>. Accessed 14 May 2022
- Mohri M, Rostamizadeh A, Talwalkar A (2018) Introduction. Foundations of machine learning, 2nd edn. MIT Press, Cambridge, pp 2–3
- Flach P (2012) The ingredients of machine learning. Machine learning—the art and science of algorithms that make sense of data, 1st edn. Cambridge University Press, Cambridge, p 14
- Lindsay GW (2020) Convolutional neural networks as a model of the visual system: past, present, and future. J Cogn Neurosci 33:1–15
- Abiyev RH, Ma'aitah MKS (2018) Deep convolutional neural networks for chest diseases detection. J Healthc Eng. <https://doi.org/10.1155/2018/4168538>
- Zou L et al (2019) A technical review of convolutional neural network-based mammographic breast cancer diagnosis. Comput Math Methods Med 2019:1–16
- Géron A (2019) Introduction to artificial neural networks with Keras. Hands-on machine learning with scikit-learn, Keras, and TensorFlow, 2nd edn. Sebastopol, O'Reilly, pp 277–291
- Neapolitan RE, Jiang X (2018) Neural networks and deep learning. Artificial intelligence—with an introduction to machine learning, 2nd edn. CRC Press, Boca Raton, pp 373–379
- Shai S-S, Shai B-D (2014) Neural networks. Understanding machine learning—from theory to algorithms. Cambridge University Press, New York, pp 228–230
- Heaton J (2015) Convolutional neural networks. Artificial intelligence for humans volume 3: deep learning and neural networks. Heaton Research Inc., Chesterfield, pp 186–194
- Raschka S, Vahid M (2017) Implementing a deep convolutional neural network using TensorFlow. Python machine learning, 2nd edn. Birmingham, Packt, pp 514–515
- Bonner A (2019) The complete beginner's guide to deep learning: convolutional neural networks and image classification. <https://towardsdatascience.com/wtf-is-image-classification-8e78a8235acb>. Accessed 30 May 2022
- Hossain A, Sajib SA (2019) Classification of image using convolutional neural network (CNN). Glob J Comp Sci Technol 19:1–7
- Lee S (2020) How to train neural networks for image classification—Part 1. <https://sandy-lee.medium.com/how-to-train-neural-networks-for-image-classification-part-1-21327fe1cc1>. Accessed 30 May 2022
- Rebala G, Ravi A, Churiwala S (2019) Machine learning definition and basics. An introduction to machine learning. Springer Press, Cham, pp 1–2
- Nandi A, Pal AK (2022) Interpreting machine learning models. Apress, Bangalore, pp 141–278
- Agarwal N, Das S (2020) Interpretable machine learning tools: a survey. In: Presented at the IEEE SSCI, pp 1528–1534. <https://doi.org/10.1109/SSCI47803.2020.9308260>
- Ribeiro MT, Singh S, Guestrin C (2016) Why should i trust you? Explaining the predictions of any classifier. arXiv preprint, pp 1–10
- Das S et al (2020) Taxonomy and survey of interpretable machine learning method. In: Presented at the IEEE SSCI, pp 670–677
- Kamath U, Liu J (2021) Explainable artificial intelligence: an introduction to interpretable machine learning. Springer Press, Cham, pp 192–224
- Biecek P, Burzykowski T (2021) Explanatory model analysis—explore, explain and examine predictive models. CRC Press, Boca Raton, pp 95–115

28. Cian D, Gemert JV, Lengyel A (2020) Evaluating the performance of the LIME and Grad-CAM explanation methods on a LEGO multi-label image classification task. arXiv preprint
29. Molnar C (2021) Model-agnostic methods. Interpretable machine learning—a guide for making black box models explainable, 2nd edn. Munich, Christoph Molnar, pp 140–178
30. Nayak A (2019) Idea behind LIME and SHAP. <https://towardsdatascience.com/idea-behind-lime-and-shap-b603d35d34eb>. Accessed 29 July 2022
31. Lundberg S, Lee S-I (2017) A unified approach to interpreting model predictions. In: Proc. ICNIP, Long Beach, CA, USA, pp 4768–4777
32. Zhang T (2021) Deep learning model interpretation using SHAP. <https://towardsdatascience.com/deep-learning-model-interpretation-using-shap-a21786e91d16>, Accessed 29 July 2022
33. Hartner R, Mezhuyev V (2022) Time series-based forecasting methods in production systems: a systematic literature review. *Int J Ind Eng Manag* 13(2):119–134. <https://doi.org/10.24867/IJIEEM-2022-2-306>
34. Hartner R, Komar J, Mezhuyev V (2022) An approach for increasing the throughput of CNN-based quality inspections systems in constrained environments. In: 11th international conference on software and computer applications (ICSCA 2022), February 24–26, 2022, Melaka, Malaysia, pp 179–184. <https://doi.org/10.1145/3524304.3524330>
35. Mezhuyev V, Gunchenko YO, Shvorov SA, Chyrchenko DV (2020) A method for planning the routes of harvesting equipment. *Autosoft. Advanced ICT and IoT technologies for the fourth industrial revolution*, vol 25
36. Hartner R, Mezhuyev V, Tschandl M, Bischof C. Data-driven digital shop floor management: a practical framework for implementation. In: ACM proceedings of the International conference ICSCA 2020, February 18–21, 2020, Langkawi, Malaysia, pp 41–45
37. Mueller C, Mezhuyev V (2022) AI models and methods in automotive manufacturing: a systematic literature review. In: Al-Emran M, Shaalan K (eds) *Recent innovations in artificial intelligence and smart applications*, vol 1061. *Studies in computational intelligence*. Springer, Cham. https://doi.org/10.1007/978-3-031-14748-7_1

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Anja Stadlhofer received a bachelor's degree in information management from FH Joanneum, Graz, Austria, and is currently studying in the master's program of international industrial management at FH Joanneum, Kapfenberg, Austria, where she focuses on the research field of machine learning. She is presently working as an IT project manager.



Vitaliy Mezhuyev Ph.D. (Educational Technology), Kyiv National Pedagogical University, Ukraine; 2002; ScD (Information Technology), Odesa National Technical University, Ukraine, 2012. Professor of informatics in Berdyansk State Pedagogical University, Ukraine, 2004–14. Professor of informatics in University Malaysia Pahang, 2014–19. From 2019 with the Institute of Industrial Management in FH Joanneum, Kapfenberg, Austria. Visiting professor at six international universities. Participated in multiple international scientific and industrial projects, devoted to the design, development, and formal verification of computer systems. Published over 140 scientific papers in peer-review journals, including highly reputed venues such as *Complexity*, *Computers & Education*, *Cybernetics and Systems*, *IEEE Access*, *Information Management*, *Intelligent automation and soft computing*, *Technology in Society* among many others.