A comparison of methods for generation of perturbed samples for LIME.

Mariana Chaves

Supervisor: Damien Garreau

Inria & Université Côte d'Azur

Contents

1 Introduction

The rapid evolution of computer vision models has given rise to complex architectures involving large amounts of parameters. Given the rising interest in applying those tools, such complexity made evident an interesting perk of machine learning: even though we possess complete knowledge about the inner works of those systems, it is infeasible for a human to directly understand them. They are mostly black-boxes lacking intrinsic or intuitive interpretability. That is, observing outputs from those models does not tell us what are the key features that determine the prediction, at least not immediately. The field of explainable machine learning intends to close this gap this gap and proposes many methods to address it such as those in Ribeiro et al. [2016], Ribeiro et al. [2018], Lundberg and Lee [2017], Simonyan et al. [2014], and Shrikumar et al. [2017], just to mention a few. Linardatos et al. [2021] extensively analyze and compares a variety of recent explainability methods.

In this project, we focus on the interpretability technique proposed in Ribeiro et al. [2016], know as Local Interpretable Model-agnostic Explanations (LIME), restricted to its application to image classification. This method is based on studying the change in the prediction probabilities caused by controlled distortions to the input image. In this work, we propose different methods to perturb images and compare their results to those given by the original version of LIME.

To be more specific, the original technique generates distorted versions of an image by randomly select some regions of the original image ξ to be changed. We call *turned off* regions the parts of the image that will suffer a change and *turned on* the ones that will not. We explore two alteration approaches for *turned off* regions. In the first one, we start by generating a replacement image ξ^*. Then we replace the *turned off* regions of the original image with the respective pixels in ξ^*, while the pixels in the *turned on* regions remain unchanged. With the default settings, LIME follows this same process to distort images, simply using the mean of the pixels in the *turned off* regions for replacement. We explore 7 other ways of creating ξ^* and, therefore, changing the way perturbed samples are generated. The second approach derives from the work of Agarwal and Nguyen [2020], where they use an inpainting model to fill the *turned off* regions.

In Section 2, we provide the terminology related to interpretability, its definition, importance, and taxonomy. Section 3 describes LIME and its internal process, as well as the details of our implementation, and examples when this technique faces difficulties to provide good explanations. The eight different perturbed samples generation methods are explained in Section 4. Subsequently, we compare the similarities of default LIME and LIME using the new perturbed samples generation methods in Section 5. While in Section 6 we evaluate the quality of the explanations of the different methods.

This study takes place as part of the Maasai team in colaboration with Université Côte d'Azur. Maasai is a research team at Inria Sophia-Antipolis, working on the models and algorithms of Artificial Intelligence. This is a joint research team with the laboratories LJAD (Mathematics, UMR 7351) and I3S (Computer Science, UMR 7271) of Université Côte d'Azur. The team focuses mainly on four axes: unsupervised learning, theory of deep learning, adaptive and robust learning, and learning with heterogeneous data. During the past years, one of its many lines of action has consisted in analyzing interpretability methods. This study comes to contribute to this field.

2 Interpretability

2.1 What is interpretability and why is it important

Before further diving into the details of LIME, let us start by clarifying interpretability and its main concepts. Interpretability is defined as the degree to which a human can understand the cause of a decision, or consistently predict a system's result. In terms of models, one is considered more interpretable than another if its decisions are easier for a human to comprehend [Molnar, 2019].

Molnar [2019] also states that the importance of interpretability lies in the fact that if the model can explain decisions we can check its fairness (ensuring that predictions are unbiased and do not implicitly or explicitly discriminate against underrepresented groups), privacy (ensuring that sensitive information in the data is protected), reliability (ensuring that small changes in the input do not lead to large changes in the prediction), causality (check that only causal relationships are picked up) and trust, as it is easier for humans to trust a system that explains its decisions. Ribeiro et al. [2016] makes this last point evident with the hypothetical example of a machine learning model used for medical diagnosis or terrorism detection, whose predictions cannot be acted upon on blind faith, since its consequences could be of high impact. Finally, there is the need to evaluate the model as a whole before deployment. Usually, models are evaluated using accuracy metrics. However, a model can provide the right answer for the wrong reasons [Garreau and Mardaoui, 2021]. Hence, inspecting individual predictions and their explanations is a worthwhile addition to accuracy metrics.

2.2 Taxonomy of interpretability methods

The different characteristics of interpretability methods allow us to categorize them. First, a technique can provide intrinsic or *post-hoc* interpretability. Intrinsic interpretability refers to models that are interpretable due to the simplicity of their structure, such as decision trees, generalized linear models (GLMs), and generalized additive models. The classic example is linear regression, where the predictions y are determined by

$$y = \beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p + \epsilon.$$

Where $x_1, \ldots, x_p$ are the features, $\beta_1, \ldots, \beta_p$ the weights or coefficients for each feature, β_0 is the intercept, and ϵ is the error, *i.e.*, the difference between the prediction and the real value. Since the learned relationships between the features and the prediction are linear, we know that an increase of one unit in the k-th feature, x_k, increases y by β_k, if we keep all the other features constant. This interpretation assumes that x_k is a numerical variable, but similar interpretations can be done for categorical ones. The key insight of this example is that it is its simple structure what makes the model intrinsically interpretable. In computer vision tasks, we rarely encounter models of such simplicity. Therefore, this type of interpretability will not be further examined in this study. *Post hoc* interpretability, on the other hand, refers to the application of interpretation methods after model training [Molnar, 2019]. Our method of interest, LIME, is an example of *post hoc* interpretability method.

Second, interpretability methods can also be classified as model-specific or model-agnostic. Model-specific ones are limited to a specific type of model. For instance, the interpretation of regression coefficients we just gave as an example is model-specific. We can not apply this interpretation to a neural network. Examples more related to computer vision would be saliency maps [Simonyan et al., 2014] and DeepLIFT [Shrikumar et al., 2017] which can only be applied to neural networks. Model-agnostic tools can be used on any machine learning model and are applied after the model has been trained (*post hoc*). These methods do not have access to model internals such as weights or structural information [Molnar, 2019]. LIME is an example of a model-agnostic technique.

Third, interpretability methods can explain an individual prediction (local interpretability) or the entire model behavior (global interpretability). Global model interpretability explains how parts of the model affect the predictions. For example, linear regression provides global interpretability, as its interpretations apply for all instances. Local interpretability aims to answer why the model made a specific prediction. LIME works with local interpretability.

3 LIME

Local Interpretable Model-agnostic Explanations (LIME) [Ribeiro et al., 2016] is a technique that explains the predictions of any classifier in an interpretable manner. The method is based on analyzing the changes to the models' predictions under distortions of the inputs. To that end, for a fixed observation, one generates a dataset consisting of perturbed samples and the corresponding predictions of the classifier. This data is then used to train a surrogate linear model using ridge regression, which is weighted by the proximity of the sampled instances to the instance of interest [Molnar, 2019].

3.1 LIME for images

LIME's detailed operation depends on the data type, that is, the process varies if we work with tabular data, text, or images. Garreau and Luxburg [2020] explains the case for tabular data, while Mardaoui and Garreau [2021] cover the details for text data. A version of the process restricted to the case of image classification, as described in Garreau and Mardaoui [2021], is explained in Algorithm 1. Let us give some additional detail for step:

1. Given an image ξ, we first partition its pixels into d contiguous patches of pixels that share color and/or brightness similarities. This process is known as *segmentation*. We call those patches *superpixels* and each pixel belongs to exactly one of them. By default, LIME uses the *quickshift* segmentation algorithm [Vedaldi and Soatto, 2008]. It generates a matrix S with the same dimensions as ξ that indicates the superpixel to which each pixel belongs (line 1 in Algorithm 1).

2. We create n new images $x_1, \dots, x_n$ by *turning the superpixels on or off*. We turn a superpixel off by replacing its pixels through some process. The default procedure is called *mean replacement*, where we replace the target pixels with the mean color of the superpixel. We further explain different replacing processes in section 4. For each x_i, we randomly determine which superpixels shall be turned on or off by sampling a binary vector z_i of d samples from a Bernoulli distribution with parameter $p = 1/2$. The 0s in the vector indicate the superpixels that should be turned off, *i.e.*, the ones that will be changed. We call the image whose superpixels are all turned off the *replacement image* and denote by ξ^*. The default number of perturbed samples n is 1000. In Algorithm 1, we can see these steps from lines 3 to 6. The replacement image ξ^* is generated in line 3. The binary vector z_i is defined in line 5. The perturbed images x_i are created in line 6, where the value of the pixel in position (k, l) will be the same as in the original image ξ if the superpixel to where it belongs is *turned on* (that is, $(z_i)_{S_{k,l}} = 1$), or it will take the value of the corresponding pixel in the replacement image ξ^* if the superpixel to where it belongs is *turned off* (that is, $(z_i)_{S_{k,l}} = 0$).

3. The classifier f is applied to the perturbed images $x_1, \dots, x_n$, leading to predictions $y_i = f(x_1)$ (line 7).

4. We measure the similarity between each of the new images x_i and the original image ξ (line 8). That is, we define a vector of weights $\pi \in \mathbb{R}^n$ by

$$\pi_i = \exp\left(\frac{-d_{\cos}(\mathbf{1}, z_i)^2}{2v^2}\right) \quad \forall u, v \in \mathbb{R}^d,$$

   where v is a positive *bandwidth parameter*, which is set to 0.25 by default, and $d_{\cos}$ is the cosine distance, defined by

$$d_{\cos}(u, v) = 1 - \frac{u^T v}{\|u\| \cdot \|v\|}.$$

5. We estimate the impact of each superpixel on the predictions of the classifier using the coefficients obtained from a ridge regression [Hoerl and Kennard, 1970] with regularization parameter $\lambda = 1$, which is the default in library Scikit-learn Python library [Pedregosa et al., 2011], where the binary vectors z_i are the features and y_i, the responses (line 10). We call this linear model the *surrogate model*. The output of this model is the set of parameters $\hat{\beta}_0, \hat{\beta}_1, \dots, \hat{\beta}_d$, where $\hat{\beta}_0$ is the intercept and $\hat{\beta}_j$ corresponds to the coefficient associated to the j-th superpixel. Positive coefficients with greater magnitudes indicate higher importance of the superpixels with respect to the model's predictions.

Let us further explain the expression $(z_i)_{S_{k,l}}$ in line 6 of the algorithm. Consider a small 4×4 image segmented into 3 superpixels. That means the values in the entries of the matrix S can be 1, 2 or 3. For instance, take

$$S = \begin{bmatrix} 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 3 \\ 1 & 2 & 2 & 3 \\ 1 & 2 & 3 & 3 \end{bmatrix}.$$

When generating the first perturbed sample, x_1, we sample the corresponding vector $z_1 \in \{0, 1\}^3$. Suppose $z_1 = (1, 0, 1)$. To define the value of the pixel $(x_1)_{1,1}$, that is, the pixel in the first row ($k = 1$) and first column($l = 1$) of the first perturbed sample (x_1), we need to know the value of $(z_1)_{S_{1,1}}$. Since $S_{1,1} = 1$, we have

$$(z_1)_{S_{1,1}} = (z_1)_1 = 1.$$

Algorithm 1: LIME for images algorithm

```
    Input: an image $\xi$; a classifier $f$; a function $g$ that generates the replacement image; the number $n$
           of perturbated images to generate; $v$ a bandwidth parameter.
    Output: the interpretable coefficients $\hat{\beta}_0, \hat{\beta}_1, \dots, \hat{\beta}_d$.
 1  $S \leftarrow \text{quickshift}(\xi)$
 2  $d \leftarrow$ the number of unique values among the entries of $S$
 3  $\xi^* \leftarrow g(\xi, S)$
 4  for $i \in \{1, 2, \dots, n\}$ do
 5      $z_i \leftarrow$ vector of $d$ i.i.d. random samples from a Bernoulli distribution with parameter $1/2$
 6      $(x_i)_{k,l} \leftarrow \begin{cases} \xi_{k,l}, & \text{if } (z_i)_{S_{k,l}} = 1 \\ \xi^*_{k,l}, & \text{if } (z_i)_{S_{k,l}} = 0 \end{cases}$
 7      $y_i \leftarrow f(x_i)$
 8      $\pi_i = e^{\frac{-d_{\cos}(1, z_i)^2}{2v^2}}$
 9  end
10  $\hat{\beta} \leftarrow \text{Ridge regression}(\text{features} = z, \text{responses} = y, \text{weights} = \pi)$
```

This means the pixel would keep the value as in the original image. Now let us define the value of $(x_1)_{1,3}$, that is, the pixel in the first row ($k = 1$) and third column($l = 3$) of the first perturbed sample (x_1). In this case $S_{1,3} = 2$, thus

$$(z_1)_{S_{1,3}} = (z_1)_2 = 0.$$

The pixel in position $(1, 3)$ would be *turned off*, *i.e.*, it would take the value of the corresponding pixel in the replacement image.

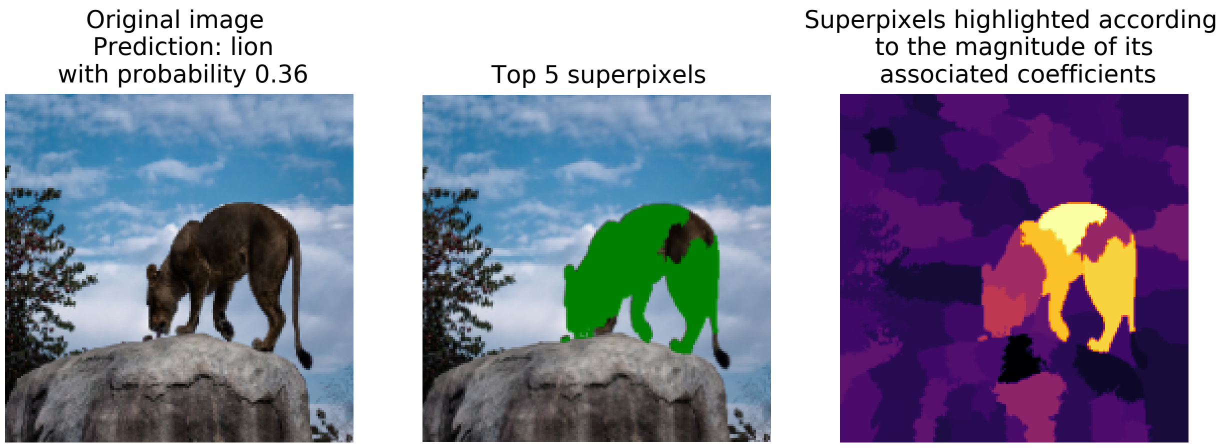


Figure 1: LIME explanations. In this example, the image ξ contains a lion. We highlight the superpixels that have the top five coefficients after running LIME with default parameters (middle panel). In the heatmap in the right panel, we color the superpixels according to the magnitude of their corresponding coefficient. These explanations intend to show the portions of the image that make the classifier f (InceptionV3) determine that the image contains a lion.

Usually, the main interest does not lie in the value of the coefficients $\hat{\beta}$ itself, but in the relationship between their magnitude and their corresponding superpixel. This is what allow us to understand which sections of the image influence the classifier's predictions the most, as we exemplify in Figure 1.

3.2 Our implementation of LIME for images

In order to experiment with different perturbed samples generation techniques, we developed our own implementation of LIME, which follows Algorithm 1. It takes inspiration from the Python implementation of LIME for images in the LIME package. The desired number of top classes defaults to only 1 but more can be selected. As in the package, our function was defined to provide results of the top classes selected by the classifier. For instance, if we consider the image in Figure 1, the first prediction of the classifier f is "lion," followed by "cougar" and "ibex." We can use our implementation to get explanations for each of these classes. That is, get an idea of the sections of the image that would lead the classifier to select "lion," "cougar" or "ibex". Additionally, we can require explanations for any specific class. This is specially useful when the true class does not figure among the top classes.

Since LIME is model-agnostic, we can choose the classifier f to be used. In this study, we employ the pre-trained model of InceptionV3 [Szegedy et al., 2016] trained on ImageNet [Russakovsky et al., 2015].

The results of our implementation include the top classes, the intercepts ($\hat{\beta}_0$), coefficients ($\hat{\beta}$) of the superpixels, the segmentation matrix, and the R-squared of the ridge model. This last metric is defined as the proportion of the variance of the response variable that is explained by the independent variables. In our case, the response variable is y (the vector of predictions done by the classifier f), and the independent variables are the binary vectors z_i.

The segmentation matrices represent most of the storing size in the results, so we opted for compressing them. For instance, the output of our implementation when using just the top 1 class comprises 702 kilobytes when the segments are not encoded, but 113 kilobytes when they are.

In Figure 2, we compare the LIME package and our implementation results for the top 7 classes predicted by the classifier InceptionV3. The first 6 most important superpixels are the same for both our function and the original one. While the seventh superpixel differs between the two functions, it corresponds to unimportant parts of the image in both of them, being part of the background.

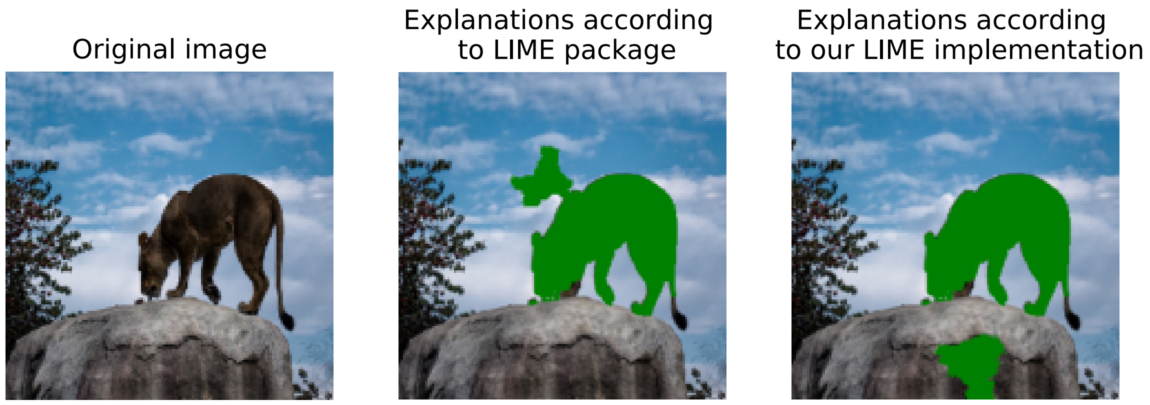

Original image
Explanations according to LIME package
Explanations according to our LIME implementation

Figure 2: Comparison of results between the LIME package and our LIME implementation using default LIME (*i.e.*, mean replacement) and 1000 perturbed samples for the most probable class according to the classifier (InceptionV3). The same results are observed between the two approaches, with the exception of the seventh most important superpixel.

If we consider only the coefficients for the most probable class ("lion"), we can observe that the values of the coefficients are very close. The intercept of obtained by the LIME package is -0.15266, while our function's is -0.15022. We observe similar behavior for the coefficients $\hat{\beta}$ (Table 1). In Figure 3 we can more easily appreciate the similarity between the two implementations in regard to the magnitude of the coefficients. The differences can be explained by the randomness associated to the generation of the perturbed images.

Both implementations take around 3 minutes in an Intel(R) Core(TM) i7-7500U CPU and 16GB of RAM to process an image using 1000 perturbed samples ($n = 1000$).

Those results indicate consistency between our implementation of LIME and the one available as a Python package. Thus, we use our implementation with default parameters as a basis for comparison when studying the new perturbed sample generation techniques.

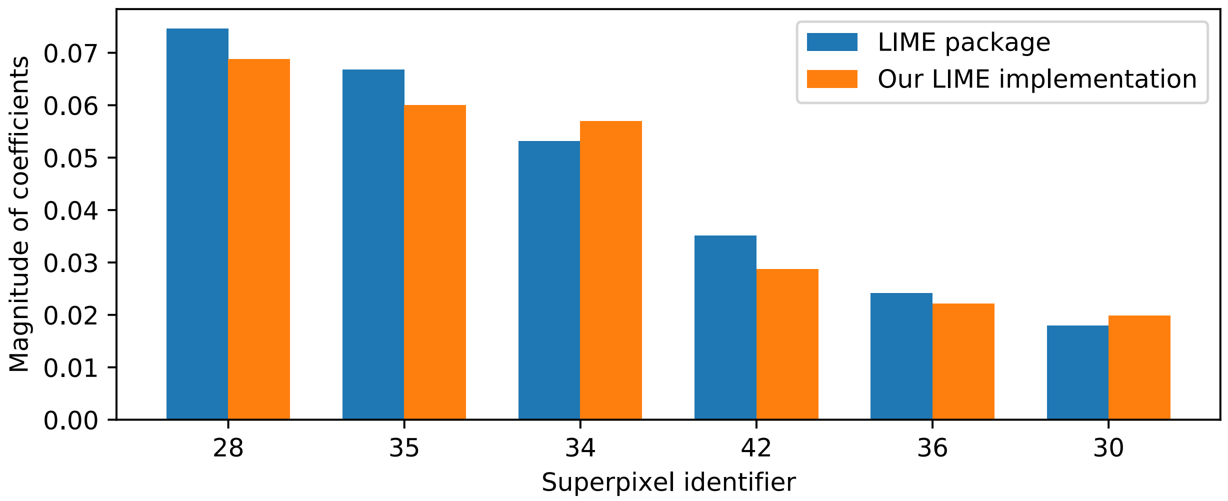


Figure 3: Comparison of results between the LIME package and our LIME implementation using default LIME (*i.e.*, mean replacement) and 1000 perturbed samples for the top 1 most probable class according to the classifier (InceptionV3). The bars show the top 6 superpixels whose coefficients have the greatest magnitudes.

Table 1: Comparison of the top 7 coefficients obtained by the LIME package and our LIME implementation for the most probable class, "lion," using mean replacement and 1000 samples. Numbers in bold indicate the superpixel identifiers that differ. Similar results are observed between the two approaches, except for the seventh superpixel.

LIME package		Our LIME function	
Superpixel	Coefficient	Superpixel	Coefficient
28	0.0746	28	0.0688
35	0.0667	35	0.0600
34	0.0532	34	0.0570
42	0.0351	42	0.0287
36	0.0241	36	0.0222
30	0.0179	30	0.0198
25	0.0149	**63**	0.0187

3.3 When default LIME fails

Garreau and Mardaoui [2021] identified images where LIME (using the default parameters) fails to provide good explanations. Images whose replacement of the most important superpixel(s) looks very similar to original(s) ones represent a challenge for LIME. Ideally, when a superpixel is *turned off* we would like that part of the image to "disappear." What we intent by making this part of the image disappear is to contrast how the classifier behaves when it "sees" that part of the image or not. If *turning off* the superpixel results in little change compared to the original image, the method does not seem to be ideal to identify if that superpixel is important or not. This situation can occur when using mean replacement, the default for LIME.

In an intend to better understand this behavior and possibly improve it, we visually inspected 5500 images from the testing set of the Imagenet challenge of 2017 [Russakovsky et al., 2015] looking for images that seemed susceptible to the effect. We opted for images from the testing set to avoid working with those that were used in the training of the model. In this visual inspection, we compare the image ξ against the replacement image ξ^* (in which all the superpixels are turned off) obtained by the *mean replacement* procedure. This allowed us to select around 100 images that we judged prone to suffer from the phenomenon of interest. Subsequently, we applied LIME to all of them to identify the failure cases.

When more examples became necessary, we opted to automate the method to detect problematic instances. The ideal algorithm would be, for each image in the dataset, to calculate the distance between the most important superpixels and their replacement. However, since we are looking for images for which LIME fails to identify the important superpixels, we cannot rely on it for this end. Thus, we approximated this metric by the Euclidean distance between the entire image ξ and its replacement image ξ^* and applied LIME to the 300 images with smaller such distances. In the end, both processes resulted in 12 images of failure cases. These examples are displayed in Section 9, Figures 20 to 31. In section (a) of those figures, we can observe the original image. In section (b), first image, we can observe the replacement image ξ^* when using default LIME (mean replacement). Subsequently, in section (c) of the Figures, we find the explanations provided by default LIME which do not highlight the most important sections of the image. These images correspond to examples of the phenomenon described before, where the replacement of the most important superpixel(s) looks very similar to the original(s) ones. In subsequent sections, we used these images to illustrate how different perturbed samples generation methods can affect the explanations.

We were also interested in exploring whether these images are challenging only for LIME or also other interpretability techniques. Therefore, we applied the Integrated Gradients method [Sundararajan et al., 2017] to these images. This technique computes the gradient of the prediction with respect to the input features (the pixels). The explanations have the same size as the input image: they assign each pixel a value that can be interpreted as the relevance of the pixel to the classification of the image. In Figure 4, we can observe that integrated gradients also have some difficulties identifying the most important pixels in the image of the planes, the daisy, the door and the jeans. In these images, the most important pixels in the heatmap do not "light up" as evidently. Nonetheless, we did not dive deeply into a comparison of different interpretability methods in this study, since it was out of our scope. However, the results from default LIME and integrated gradients indicate that this group of images could indeed be difficult for different interpretability methods.

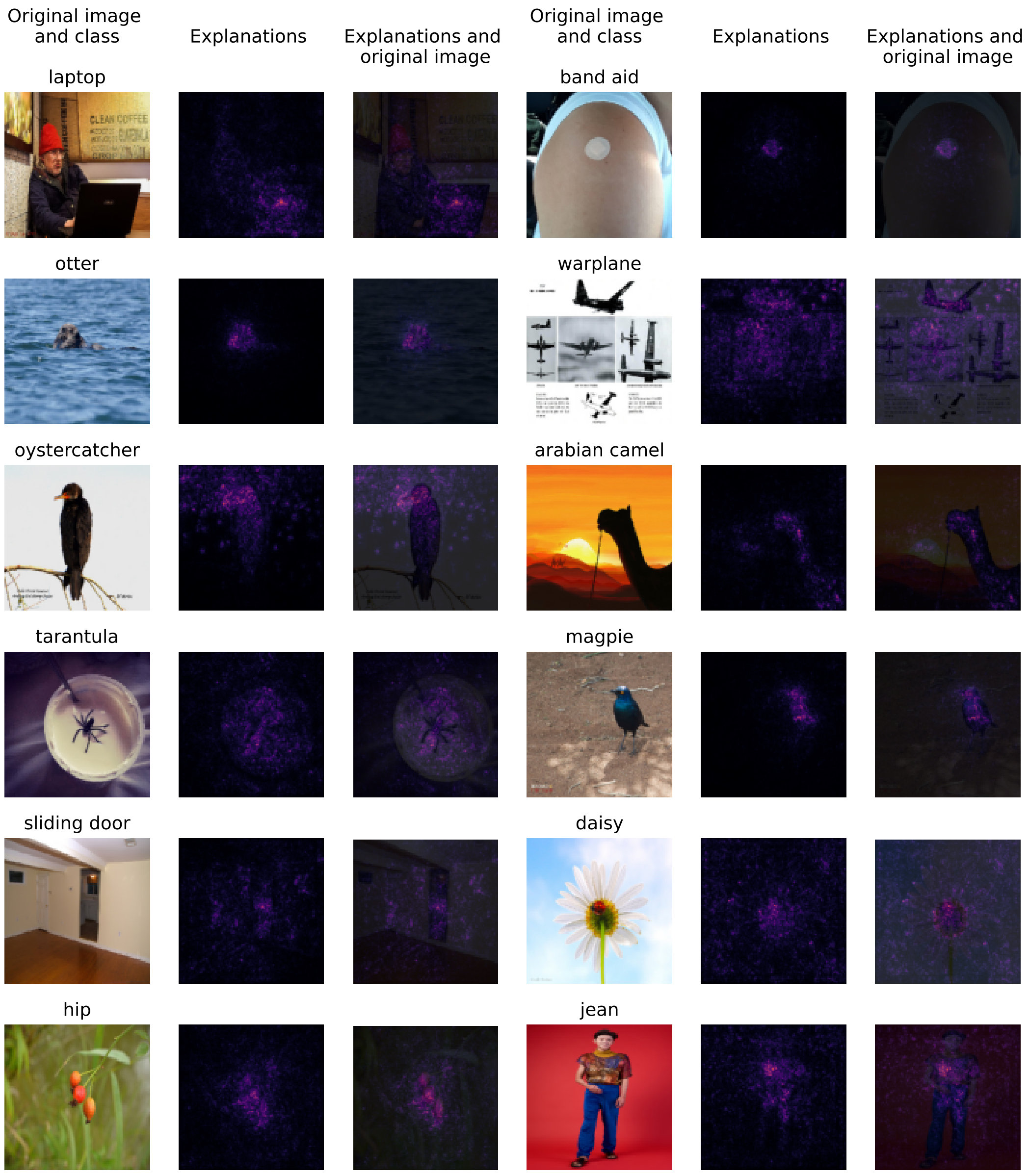


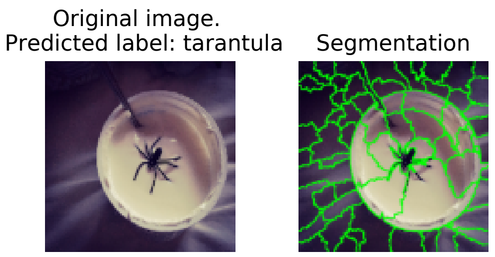
Figure 4: Explanations according to integrated gradients for a group of images where default LIME fails. From left to right for each group of images: Original image and its predicted label, integrated gradients heatmap, and the integrated gradients heatmap overlapped with the original image. Brighter colors in the heatmap identify the parts of the image that are more important for the classifier's (Inceptionv3) predictions.

4 Perturbed samples generation methods

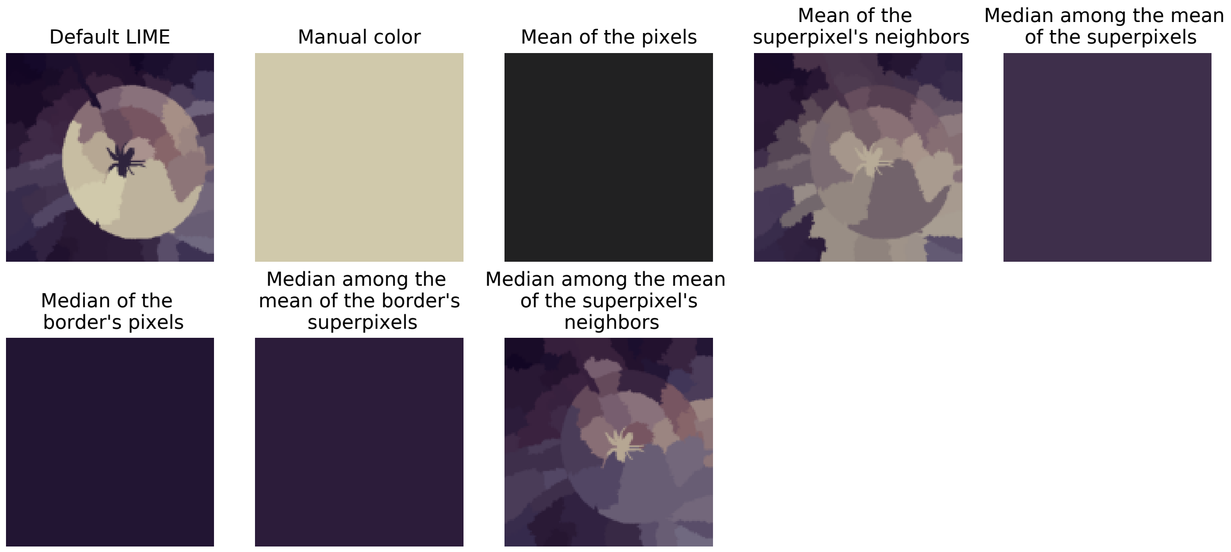
As described before in section 3.1, one of LIME's steps generates a new dataset consisting of perturbed samples $x_1, \ldots, x_n$. Those images preserve some superpixels as in the original image ξ, while others are replaced (*turned off*) by pixels from a replacement image ξ^*. Default LIME creates ξ^* by replacing the pixels by the mean color of the superpixel to which they belong.

In this section, we present new ways to generate ξ^*. We call these *replacement methods*. Subsequently, we study using inpainting models to fill the superpixels that are *turned off*. This kind of approach was first proposed by Agarwal and Nguyen [2020], under the name "LIME-G", which stands for "Generative LIME". We perform experiments using default LIME and compare them with LIME-G (Subsection 4.2) and the alternative replacement methods in Section 4.1.

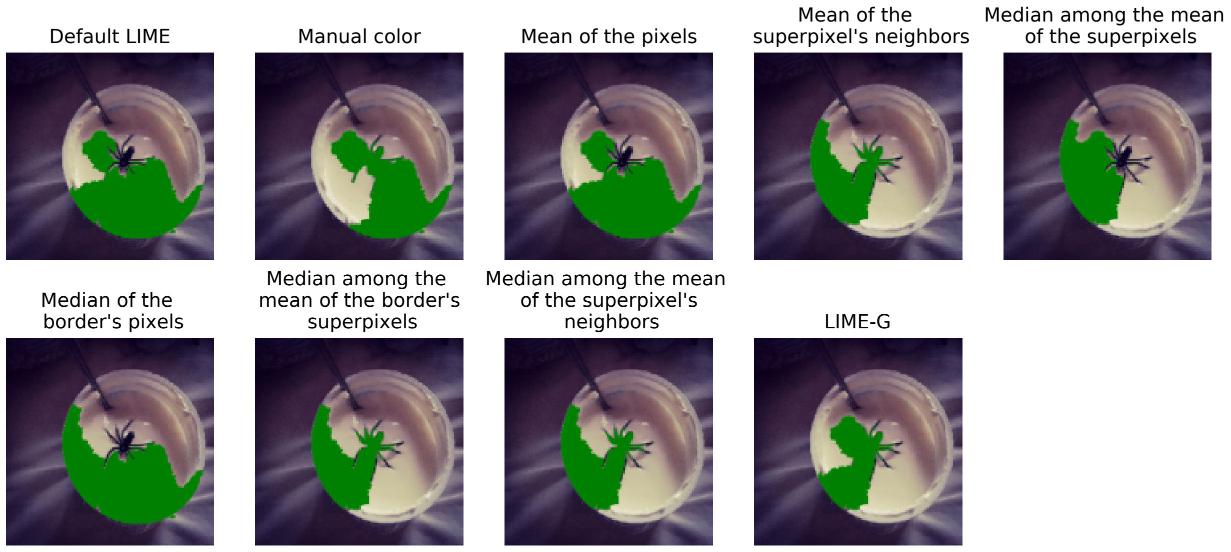
Let us provide a glance at this comparison in Figure 5. It shows the results of the different perturbed samples generation methods applied to an image of a tarantula. This is one of the cases where default LIME fails to identify the most important section of the image. In the first part of this figure (5a), we observe the original image and its segmentation. The tarantula is contained in a single superpixel, so we would like the explanations to indicate this superpixel as the most important one in the image. In the second part of the figure (5b), we observe possibilities for the replacement image according to different perturbed samples generation methods. There is no replacement related to LIME-G since it does not use any. In the third part of the figure (5c), we show the explanations provided by different methods. The first one, default LIME, does not highlight the superpixel containing the tarantula while five of the other methods do. At least for these particular case, deviating from LIME's default behavior is advantageous. Similar images can be observed from Figure 20 to 31. These correspond to the comparison of the perturbed sample generation methods applied to failure cases for default LIME.



(a) Original image and segmentation


Default LIME
Manual color
Mean of the pixels
Mean of the superpixel's neighbors
Median among the mean of the superpixels
Median of the border's pixels
Median among the mean of the border's superpixels
Median among the mean of the superpixel's neighbors

(b) Replacement images


Default LIME
Manual color
Mean of the pixels
Mean of the superpixel's neighbors
Median among the mean of the superpixels
Median of the border's pixels
Median among the mean of the border's superpixels
Median among the mean of the superpixel's neighbors
LIME-G

(c) Explanations

Figure 5: Comparison of LIME using different perturbed samples generation methods on an image of a tarantula. Original image, superpixel segmentation, and replacement image (if applicable) and explanations (top 3 superpixels) for each perturbed samples generation method.

4.1 Replacement methods

We explore, in this subsection, seven new possibilities to generate the replacement image ξ^*. Some of these processes consist in picking a single color and using it to color the superpixels that are *turned off* in the perturbed samples $x_1, \ldots, x_n$. For instance, in Figure 5b some of the replacement images consist in just one color. Other methods are superpixel-specific. That is, they select a color for each superpixel in ξ^*. All of these methods are deterministic processes, while the perturbed samples generation technique that we describe later in section 4.2 is not.

4.1.1 Replacement image based on manually picked colors

Ideally, we would generate perturbed samples by "erasing" the *turned off* sections of the input so that we know how the classifier reacts to the omission of portions of the image. However, if we observe the replacement image for default LIME in Figure 5b we see that, although the details were removed, the idea of a spider is still evident. Hence, we start with a method inspired how a human would erase important portions of the image.

Observe once more the example on Figure 5a. If we were to select a single color to make the tarantula disappear from the image, we would pick a shade of beige similar to the one that surrounds it. We used this idea to manually pick a replacement color for each of the 12 images where default LIME fails. We call this method *manual color replacement*.

In the second image of Figure 5c, we get good explanations: the superpixel that contains the tarantula is selected as the most important. The results of LIME using this replacement for the other 11 images can be observe from Figure 20 to 31. For all the 12 images, except one, this replacement identifies the crucial superpixels. The technique fails for the band aid case in Figure 21. In this image, the band aid belongs to just one superpixel, as in the tarantula example. Nevertheless, it differs from it in the fact that cannot really "erase" the band aid just by assigning a color to the superpixel. No matter the color we assign, it would still look like a band aid. Although in practice, choosing manually the replacement color is unpractical and extreme time consuming when we desire to apply the technique to many images, these results suggest that replacing the most important superpixels by a color or pattern similar to the background can improve the results of LIME.

4.1.2 Replacement image based on mean color of the pixels

Subsequently, we considered using the mean color of the image as replacement. That is, we take the average value of each of the image's channels to compose the resulting color. We call this method *mean of the pixels replacement*. The resulting color is usually a shade of gray, as it can be observed in the replacement image for *mean of the pixels* in Figures 20 to 31. This occurs because at the center of the RGB color space we find different tones of gray. This replacement fails for 3 of the 12 images considered. As we will further discuss in the following sections, it is outperformed by most of the other methods.

4.1.3 Replacement image based on the mean color of neighboring superpixels

Following the logic of the *manual color replacement*, we would like to "erase" the important section of the image. The method discussed in this subsection aims to approximate the color that we selected manually in subsection 4.1.1. That is, we look at the colors that surround a superpixel and fill it based on such colors. In practice, we do this by using the average color of neighboring superpixels (those that share a border with the target superpixel). We call this method *mean of the superpixel's neighbors replacement*.

We illustrate the previous intuition in Figure 6. Let us assume that the only superpixel *turned off* is the one containing the tarantula. Default LIME would color it with the mean of the colors inside the superpixel, which makes it remain black, as we can observe in the second image of the figure. On the third image in the figure, the tarantula's superpixel was replaced by the mean of the superpixel's neighbors, almost erasing it from the image. Furthermore, the top 5 predictions by the classifier (InceptionV3) were calculated for the 3 images. For the original image the model's top predictions are "tarantula," "petri dish," "wolf spider," "spindle," and "barn spider." Using the mean of the superpixel as replacement the predictions were "petri dish," "spindle," "syringe," "tarantula," and "bucket," so we still get tarantula among the top 5. This is

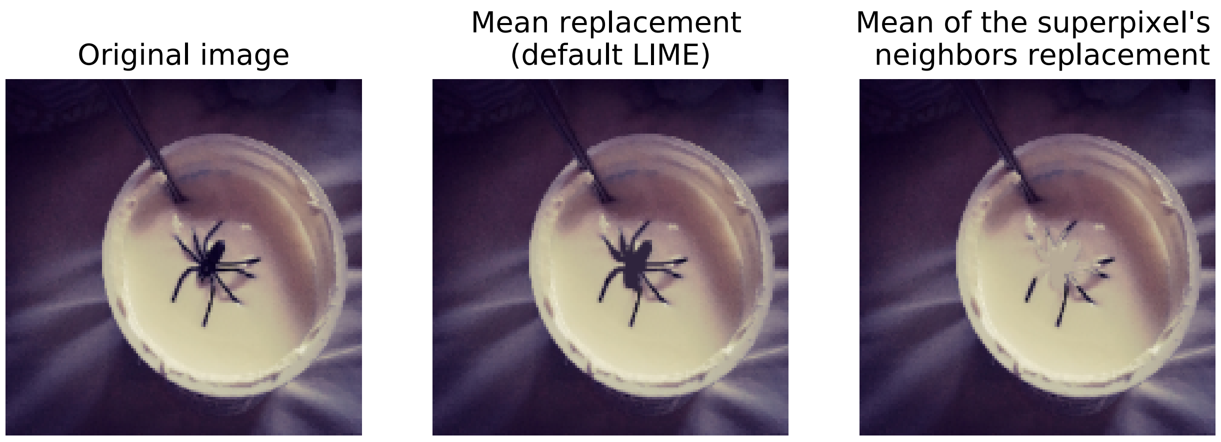

Original image
Mean replacement (default LIME)
Mean of the superpixel's neighbors replacement

Figure 6: Using mean replacement (default LIME) and mean of the superpixel's neighbors in an image where only the most important superpixel (the one containing the tarantula) is *turned off*. The original image is displayed on the left. The center image shows the effect of using mean replacement (default for LIME) for a single superpixel, resulting in little to no change compared to the original. The image on the right shows how using mean of the superpixel's neighbors replacement manages to make the tarantula mostly "disappear."

not the case for the *mean of the superpixel's neighbors replacement*. Its top 5 predictions do not include any classes related to spiders: "syringe," "petri dish," "soup bowl," "spindle" and "ladle."

To understand this replacement method, consider the first row of images in Figure 7 and Algorithm 2. In the first image we observe the 10 superpixels in which the image could be divided. For each superpixel, we go through a process of identifying the superpixel, dilating it to recognizing its outer edge, and identifying its neighbors. Let us consider this process for the superpixel colored in yellow.

Algorithm 2: Replacement by mean of the superpixel's neighbors

```
   Input: an image $\xi$; a segmentation $S$ of $\xi$ into superpixels; the number $d$ of superpixels in $S$.
   Output: a replacement image $\xi^*$
1  $\xi^* \leftarrow \xi$
2  for $s \in \{1, 2, \dots, d\}$ do
3      $S^*_{i,j} \leftarrow \mathbb{1}_{S_{i,j}=s}$
4      $D \leftarrow \text{maxpool}_{3,3}(S^*)$
5      $B \leftarrow D - S^*$
6      $B^* \leftarrow B \odot S$
7      Let $A$ be the set of non-zero values among the entries of $B^*$
8      $N \leftarrow \begin{cases} 1, & \text{if } S_{i,j} \in A \\ 0, & \text{otherwise.} \end{cases}$
9      $N^* \leftarrow N \odot \xi$
10     $\mu \leftarrow \frac{\sum_i \sum_j N^*_{i,j}}{\sum_i \sum_j N_{i,j}}$
11     $\xi^*_{i,j} \leftarrow \begin{cases} \mu, & \text{if } S^*_{i,j} = s \\ \xi^*_{i,j}, & \text{otherwise.} \end{cases}$
12 end
```

1. The superpixel is identified using a Boolean matrix S^* that indicates whether a pixel belongs to the superpixel (line 3 in Algorithm 2). We can visualize this in the second image where the superpixel is presented in white.

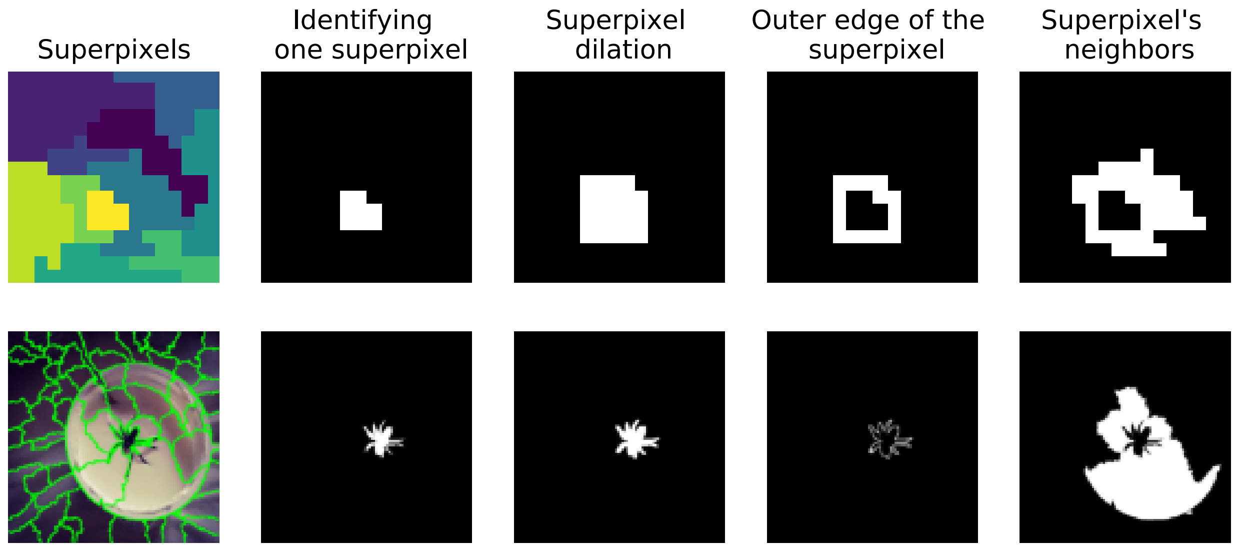


Figure 7: Superpixel's neighbors identification. First row: example in a simplified image of 15 × 15 pixels, divided into 10 superpixels. Second row: example in a image where default LIME fails (299 × 299 pixels).

2. The border of the superpixel is dilated using maxpooling (line 4). That is, a kernel slides through the boolean matrix S^* (as in 2D convolution). An entry in the matrix D will result in a 1 if at least one of the pixels under the kernel is 1. This dilates the region of 1s in the matrix, as it is presented in the third image. More formally, and following the notation in algorithm 2, this maxpooling process with a 3 × 3 kernel can be denoted as

$$C = S^* \circledast \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$D_{i,j} = \begin{cases} 1, & \text{if } C_{i,j} \geq 1 \\ 0, & \text{otherwise.} \end{cases}$$

   Where $\circledast$ denotes the convolution operator.

3. We obtain the outer edge B of the superpixel by subtracting the boolean matrix S^* that identifies the superpixel from the dilated matrix D (line 5). This gives us the forth column in the figure.

4. We identify the neighbors by recognizing the superpixels that overlap with the outer edge B (lines 6, 7 and 8). These are highlighted in the fifth column. Note that the operator $\odot$ in line 6 denotes the Hadamard product, which is the entrywise product of matrices.

5. Finally, the average color of the neighbors is computed and assigned to the superpixel in the replacement image ξ^*. (lines 9, 10 and 11)

Given that the dilation process slides a 3 × 3 kernel through the Boolean matrix S^*, and that we have an iterable process over the number of superpixels d, the computational complexity depends on the dimensions (w, the width, and h the height) of the image ξ and d:

$$\mathcal{O}(w \cdot h \cdot d).$$

In the second row of images in Figure 7, we can observe the example for one of the images where default LIME fails.

Moreover, Figure 5b displays the replacement image for *mean of the superpixel's neighbors*. In contrast to the one for default LIME, now the replacement image seems to blur the original.

We applied this replacement method to the 12 images where default LIME fails and saw that the detection of the correct superpixels improved for almost all the examples, except by two cases. One of them is the band aid image, which, as mentioned before, is more specially challenging.

This replacement method encloses a disadvantage. If the neighboring superpixels contain colors that are on different sides of the RGB space, the mean of those neighbors will be a shade of gray. For instance, suppose that half of the neighboring superpixels present shades of blue, while the other half present shades of yellow. These are opposite colors in the RGB space, so their mean will be a shade of gray. Using that shade of gray will not help "erasing" the superpixel since is does not resembles the superpixels that surround it. This problem is similar to the one presented by the *mean of the pixels replacement.*

4.1.4 Replacement image based on median color

Algorithm 3: Median among the mean of the superpixels replacement.

```
Input: an image $\xi$; a segmentation $S$ of $\xi$ into superpixels; the number $d$ of superpixels in $S$.
Output: a replacement image $\xi^*$
1 $\xi^* \leftarrow \xi$
2 for $k \in \{1, 2, \ldots, d\}$ do
3     $C_k \leftarrow \frac{\sum_i \sum_j \xi_{i,j} \cdot \mathbb{1}_{S_{i,j}=k}}{\sum_i \sum_j \mathbb{1}_{S_{i,j}=k}}$
4 end
5 $c^* \leftarrow \underset{j}{\operatorname{argmin}} \sum_i \|C_j - C_i\|_2$
6 $\xi^*_{i,j} \leftarrow c^*$
```

Given that using the mean, either of all pixels or of the neighboring superpixels, can easily drag us to the center of the RGB space, we experimented with replacing it with the median. This is an approximation of the most common color in the image, that is used to simulate the background color without recurring to more complex, and computationally costly, models, such as background detection algorithms. For instance, let us consider the image of an otter in Figure 8. The most common color is a shade of blue, which matches the color of the background, but not the otter. Using a shade of blue will keep a *turned off* superpixel in the background similar to the original, while a *turned off* superpixel in the otter will make it look like part of the background.

To explain the approximation of the central color, consider Figure 8 once again. In the first scatter plot we observe all the pixels of the image plotted according to their RGB coordinates. The image contains mainly different shades of blue and some shades of gray. We select a shade of blue by computing the geometric median of those colors: the one minimizing the sum of Euclidean distances to the other colors in the image. That is, we compute

$$\underset{j}{\operatorname{argmin}} \sum_i \|C_j - C_i\|_2,$$

where $C_1, \ldots, C_m$ are vectors of three entries representing the colors of the pixels in RGB ($C_i \in \mathbb{R}^3$). We call the geometric mean of the pixels the *median color.*

These images have a size of 299 × 299 pixels. Therefore, to calculate the median color of all pixels, $299^4 \approx 8.10^9$ distances must be computed. This takes 1 hour and 21 minutes on an Intel(R) Core(TM) i7-7500U CPU and 16GB of RAM. We can speed this up dramatically by instead computing the median of the mean color of each superpixel, which provides almost the same result as the median color of all pixels.

In Figure 8, we observe in the top right the image where we color each superpixel using the mean of the pixels it contains. In this image, there are d superpixels and, therefore, d colors. This colors are plotted in the scatter plot in the bottom right. We can obtain their median color by computing only d^2 distances, which takes 0.534 seconds. The RGB value of the median color obtained using all the pixels was (100, 139, 182), while using the mean of the superpixels gave the value (100, 139, 180).

In Figure 9, we compare the median color of the pixels and the median color among the mean of each superpixel for 6 images. These 6 images are part of the 12 images where default LIME fails, each identified

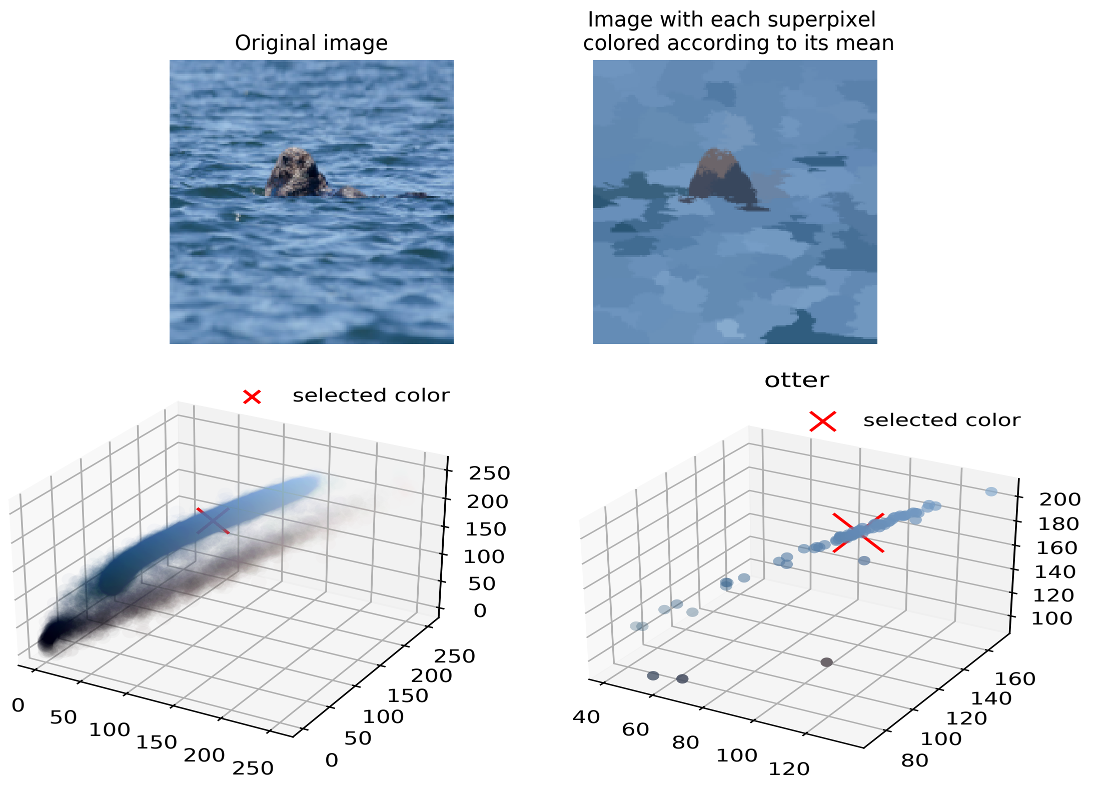


Figure 8: On the top left, we observe an image of 299 × 299 pixels. Hence, this image contains 299^4 colors which are displayed in the RGB space in the scatter plot on the bottom left. On the top right, we show an image where each superpixel was colored according to its mean. Note that this image corresponds to the replacement image generated when using default LIME. This image contains $d = 82$ superpixels, therefore we take into consideration 82 colors, which are displayed in the scatter plot on the bottom right. The median color in each scatter plot is indicated with a red "X." The two median colors are almost identical.

by the class to which it belongs. We can observe that the two colors are very similar in all cases, indicating it is possible to approximate the real median color by the median color among the mean of the superpixels.

For now on, we use superpixels' mean color to compute the median color. We call using this median color as replacement color *median among the mean of the superpixels replacement.* In Algorithm 3, lines 2 to 4 show the process to get a vector C that contains the mean color for each superpixel. In line 5, the geometric median is applied to get the median color c^*.

In Figure 10, we plot the mean color of the superpixels for each of the 12 images, and indicate the selected color (median) that is used as replacement. From Figure 20 to 31, we see the explanations using this type of replacement. This technique improves the results compared to default LIME for 9 of the 12 images.

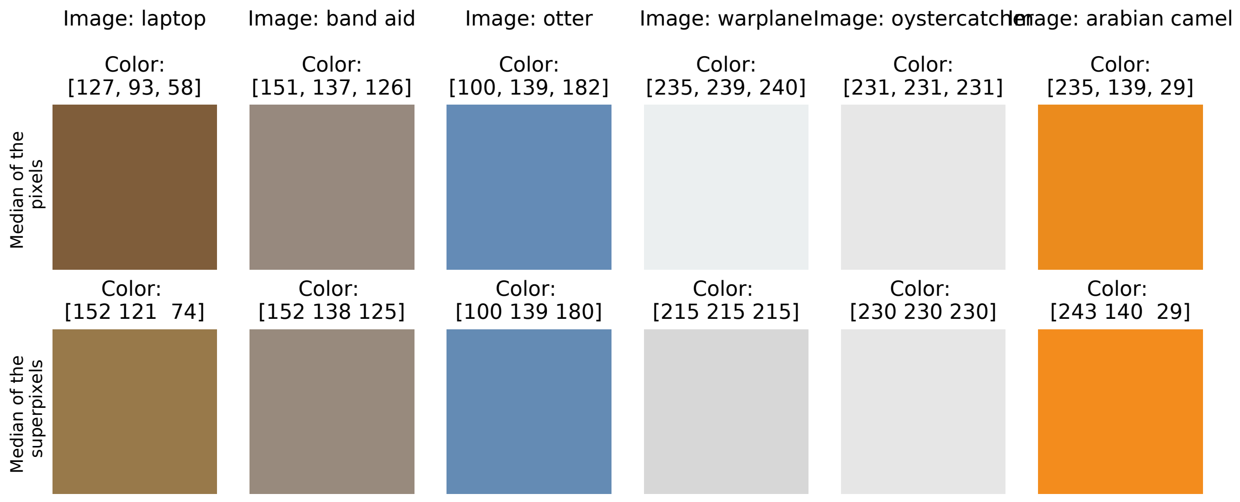


Figure 9: Comparison between the median color of the pixels and the median color among the mean of the superpixels for 6 images. The results of both techniques are very similar, which indicates that using the median color among the mean of the superpixels is an effective approximation of the median color of the pixels. Given its lower computational cost, we prefer using he median color among the mean of the superpixels.

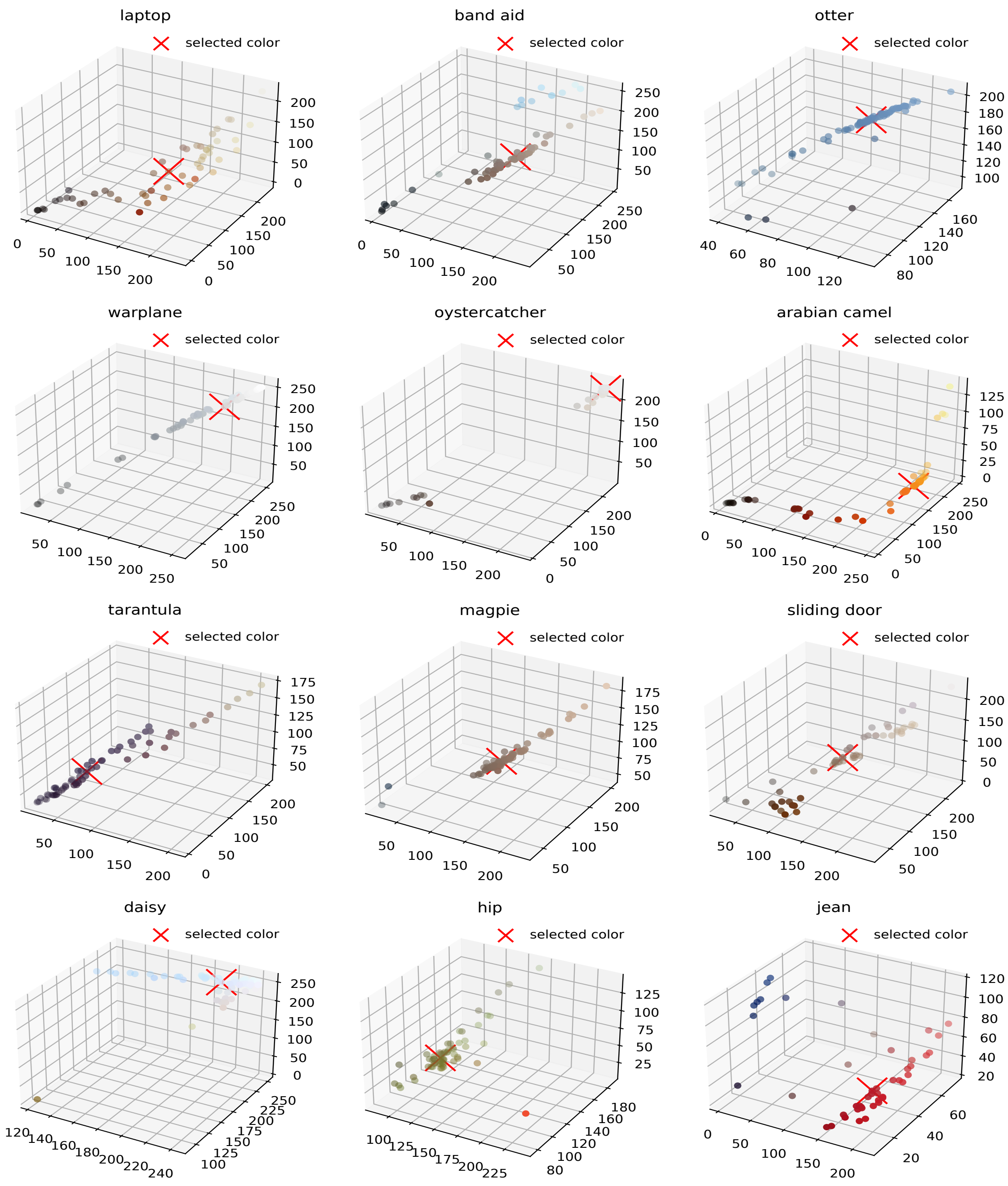


Figure 10: Median color of the superpixels in the RGB space for the group of images in which default LIME fails. The selected color using geometric median is denoted with an x.

4.1.5 Replacement image based on the median color of the pixels in the border

Next, we considered taking the median color among the pixels in the border of the image, as those rarely touch the main object (Algorithm 4). We call this replacement *median color of the border's pixels.* We can observe its results in figures 20 to 31. This replacement achieves better results than default LIME for 10 of the 12 images.

This technique can be problematic when the border do not capture accurately the background, such as when the image includes a frame, or the colors surrounding the most important superpixels. That is the case of the band aid image (Figure 21).

Algorithm 4: Median of the border's pixels replacement.

```
   Input: an image $\xi$.
   Output: a replacement image $\xi^*$.
1  $\xi^* \leftarrow \xi$
2  $C \leftarrow$ the set of values among the entries of $\xi$ that belong to the border of $\xi$
3  $c^* \leftarrow \underset{j}{\operatorname{argmin}} \sum_i \|C_j - C_i\|_2$
4  $\xi^*_{i,j} \leftarrow c^*$
```

4.1.6 Replacement image based on the median color among the mean of the superpixels in the border

Subsequently, we tried to capture the color of the background computing the median color among the mean of the superpixels at the border of the image. This replacement method can be useful for images that have "anomalies" in their borders, like frames. This technique outperforms default LIME in 11 out of the 12 images. The exception is, again, the band aid case.

In Algorithm 5, lines 2 to 4 get the set A of superpixels that are in the border of the image. Lines 5 to 7 generate the vector C that contains the mean for each superpixel in the border. Line 8 defines the median color c^* among the colors in C.

We note that for all cases, the colors picked using the median among the mean of the superpixels, the median of the border's pixels, and the median among the mean of the border's superpixels are similar.

Algorithm 5: Median among the mean of the border's superpixels replacement.

```
   Input: an image $\xi$; a segmentation $S$ of $\xi$ into superpixels.
   Output: a replacement image $\xi^*$
1  $\xi^* \leftarrow \xi$
2  $B \leftarrow$ the set of values among the entries of $\xi$ that belong to the border of $\xi$
3  $S^*_{i,j} \leftarrow S_{i,j} \cdot \mathbb{1}_{I_{i,j} \in B}$
4  $A \leftarrow$ the set of non-zero values among the entries of S*
5  for $k \in A$ do
6      $C_k \leftarrow \frac{\sum_i \sum_j \xi_{i,j} \cdot \mathbb{1}_{S_{i,j}=k}}{\sum_i \sum_j \mathbb{1}_{S_{i,j}=k}}$
7  end
8  $c^* \leftarrow \underset{j}{\operatorname{argmin}} \sum_i \|C_j - C_i\|_2$
9  $\xi^*_{i,j} \leftarrow c^*$
```

4.1.7 Replacement image based on the median color among the mean of the neighboring superpixels

Finally, we considered a hybrid technique between the *mean of the superpixel's neighbors replacement* and the *median among the mean of the superpixels.* We identify the neighbors of each superpixel and compute the mean color of each of them. If a superpixel has b neighbors, we calculate the median color among b average colors to use it as replacement. In Algorithm 6, lines 1 to 7 are the same as in Algorithm 2, *i.e.*, they help us define the set A that contains the neighbors of a superpixel. Lines 8 to 10 create the vector C that contains the mean color of each neighbor, line 11 defines the median color c^* among the neighbors, and line 12 assigns this color to the superpixel s.

Algorithm 6: Replacement by the median among the mean of the superpixel's neighbors

```
Input: an image $\xi$; a segmentation $S$ of $\xi$ into superpixels; the number $d$ of superpixels in $S$.
Output: a replacement image $\xi^*$
1  $\xi^* \leftarrow \xi$
2  for $s \in \{1, 2, \dots, d\}$ do
3      $S^*_{i,j} \leftarrow \mathbb{1}_{S_{i,j}=s}$
4      $D \leftarrow \text{maxpool}_{3,3}(S^*)$
5      $B \leftarrow D - S^*$
6      $B^* \leftarrow B \odot S$
7      Let $A$ be the set of non-zero values among the entries of $B^*$
8      for $k \in A$ do
9          $C_k \leftarrow \frac{\sum_i \sum_j \xi_{i,j} \cdot \mathbb{1}_{S_{i,j}=k}}{\sum_i \sum_j \mathbb{1}_{S_{i,j}=k}}$
10     end
11     $c^* \leftarrow \underset{j}{\text{argmin}} \sum_i \|C_j - C_i\|_2$
12     $\xi^*_{i,j} \leftarrow \begin{cases} c^*, & \text{if } S^*_{i,j} = s \\ \xi^*_{i,j}, & \text{otherwise.} \end{cases}$
13 end
```

Once more, from Figure 20 to 31 we can observe the explanations that are obtained using this type of replacement. It clearly outperforms default LIME in all 12 images, failing in the band aid example only, as we have observe with multiple replacements.

After visually comparing the results of the different replacement methods in this set of 12 images, we considered that the *median among the mean of the superpixel's neighbors replacement* provides the best results. This replacement method and *median among the mean of the border's superpixels* provide good results for 11 of the 12 images. Nonetheless, we prefer *median among the mean of the superpixel's neighbors replacement* because it is not biased by anomalies at the borders of the image. This replacement matches the performance of manually picking the replacement color, which is clearly not feasible in large scale. For this reason, we focus on the *median among the mean of the superpixel's neighbors replacement* in further sections.

Given that, so far, we have evaluated the results only on a small set of challenging images, we might question whether this replacement method would also work on "regular" images. We take a large sample of images into consideration in further sections.

4.2 LIME-G

Interpretability methods that use perturbed samples, such as LIME, approximate the importance of a region in the image by the change in the classification probability when that region is absent *i.e.* removed from the image. These methods "erase" the region by replacing it with (a) mean pixels (which is the case of default LIME), (b) random noise, or (c) blurred versions of the original content. Agarwal and Nguyen [2020] argue that these removal techniques produce unrealistic images. Moreover, image classifiers are often easily fooled by these unusual input patterns. Hence, interpretability methods that use such removal techniques generate this type of pattern which can lead to explanations that are unreliable. They tackle this issue by using a generative inpainting model to color the pixels that are *turned off* in each perturbed sample x_i and, therefore, generate more realistic images. They implemented this approach on various interpretability methods, including LIME. They call LIME-G to the modified version of LIME using generative inpainting

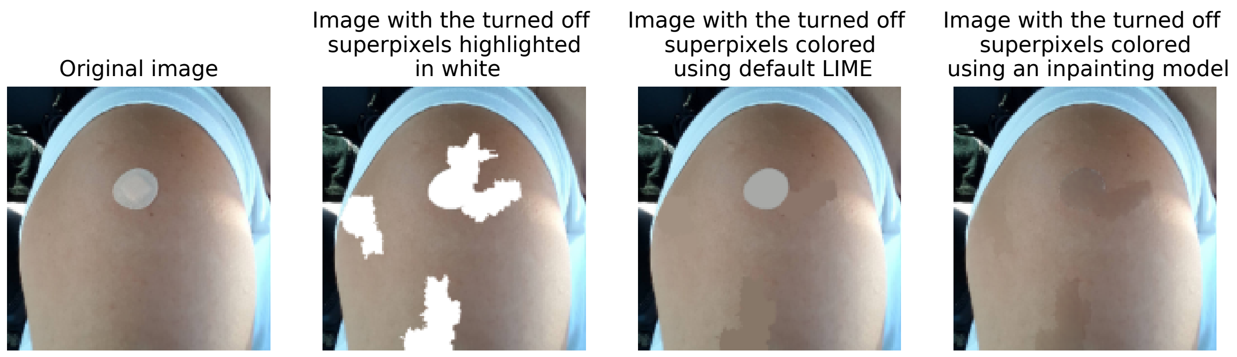

Original image
Image with the turned off superpixels highlighted in white
Image with the turned off superpixels colored using default LIME
Image with the turned off superpixels colored using an inpainting model

Figure 11: Example of perturbed sample where five superpixels are *turned off*. The original image is shown at the left. The second image highlights in white the five *turned off* superpixels. In the third one, the superpixels are *turned off* using default LIME, while in the forth the superpixels are *turned off* using an inpainting model (LIME-G).

models.

The *Image Inpainting* problem consists of, given an image with some portion missing, to fill the absent region in a seamless, plausible manner. Notice the problem is, in principle, different from image reconstruction. For instance, if the missing region was obtained by erasing pixels from a source image, it is not needed to recover the original image in order to have a good inpainting.

To illustrate the practicality of inpainting models, consider the band aid example (Figure 11, first image). As it was mentioned before, the band aid case resulted challenging for the replacement methods in Section 4.1 due to the fact that no matter what color is applied to the superpixel that contains the band aid, the image still resembles a band aid. Let us imagine that to generate one of the perturbed images x_i five of the superpixels are *turned off*. One of them being the one that contains the band aid. In Figure 11, in the second image, we highlight in white the 5 superpixels that are *turned off*. If we use default LIME, the perturbed sample would look like in the third image. We can still recognize the band aid. If, instead, we use an inpainting model the perturbed sample would look like in the fourth image. The band aid is no longer obvious.

Agarwal and Nguyen [2020] used Deepfill V1 [Yu et al., 2018] as inpainting model in their version of LIME-G. This inpainting model works properly only when the region to fill has a rectangular shape. In their case they were using the *SLIC* segmentation [Achanta et al., 2012] algorithm, which, under the parameters they set, produces rectangular-like superpixels. As we have shown, *quickshift* segmentation produces irregular-shaped superpixels. Hence, to implement the LIME-G approach, we explored other inpainting models that deal better with big irregular areas to inpaint, such as the ones proposed by Zhao et al. [2021], Li et al. [2020a], Liu et al. [2020], Wadhwa et al. [2021], and Li et al. [2020b]. We also explored a more classic inpainting technique by Telea [2004]. In all cases, these models take as input a color image ξ and a binary mask that determines which regions should be inpainted. The binary mask is a 2D matrix with its dimensions corresponding to the width and height of the image ξ.

Consider the lion image we presented in Figure 1. To generate perturbed samples from it, we need masks like the ones presented in the first row of Figure 12. The white parts of the mask determine que region to be inpainted, and therefore, ideally "erased." In the fifth column, we also include a mask that indicates the model to inpaint the superpixels that contain the lion, this with the idea of making the lion completely disappear. In the same figure, we show the inpainting results using three inpainting techniques: TELEA [Telea, 2004], Deepfill V2 [Yu et al., 2018], and DeepGIN [Li et al., 2020b]. Note that TELEA, manages to erase the parts of the lion indicated by the mask, but it creates some convex polygon-shaped sections all across the image. Deepfill V2 reconstructs the lion instead of erasing it. Additionally, it "borrows" some patterns from sections of the image to replicate them in places where they do not fit that well. For instance, we observe that, in some of the results, it uses the pattern of the tree to fill the lion. DeepGIN succeeds on erasing the masked parts of the lion, with the disadvantage of creating blurred sections. For the mask that considers only the superpixels that contain the lion, we see the lion disappear almost completely. None of

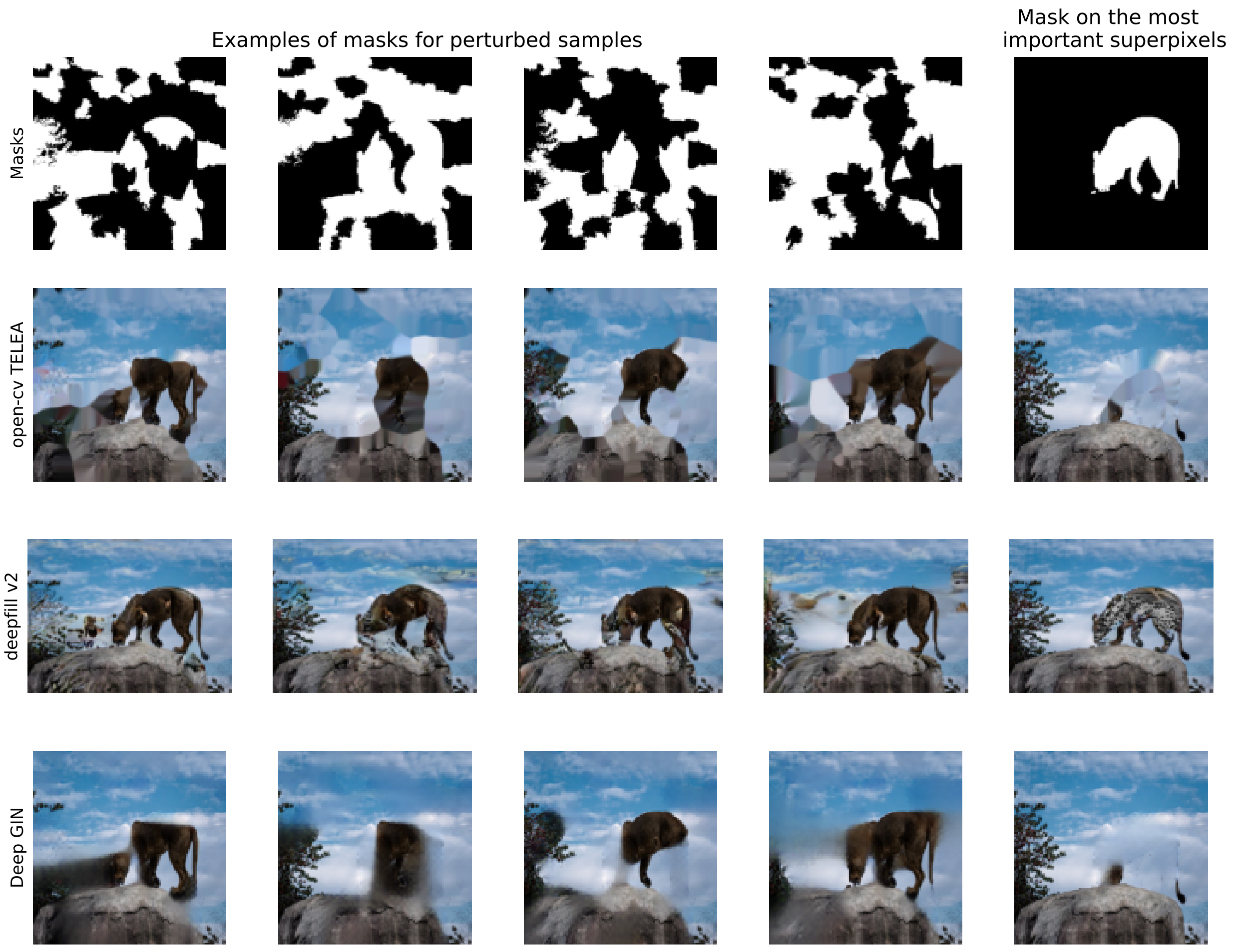

Examples of masks for perturbed samples
Mask on the most important superpixels
Masks
open-cv TELEA
deepfill v2
Deep GIN

Figure 12: Comparison of three generative inpanting models on the image of a lion using different masks. The first row presents the masks, where the white parts determine the region of the image to be inpainted. The first 4 exemplify the type of masks that LIME-G uses to create perturbed samples. In the last column, the mask determines to inpaint the superpixels that contain the lion. Rows 2 to 4 show the results of the different inpainting techniques. DeepGIN provides the best results given that it succeeds on erasing the parts of the lion that are masked and replacing them with credible backgrounds.

the inpainting techniques results in images as realistic as desired, but among them, DeepGIN provided the best results. Hence, we used this model in our version of LIME-G.

DeepGIN was designed for "extreme inpainting" scenarios. This means that it was trained to fill proportionally big, arbitrarily shaped regions of the images. In fact, it took a part in AIM 2020 ECCV Extreme Image Inpainting Challenge [Li et al., 2020b] DeepGIN is trained under an adversarial setup: while training the generative network, two discriminatory models were trained simultaneously to detect imprecisions in the generative process. Those discriminatory models are only used for training while the generative model output is the inpainted images. This is done in two main stages. First, a coarse inpainting is generated, then the rest of the network add texture and details to this first approximation to achieve seamlessness.

Note that the use of inpainting models carries a cost in execution time. For instance, running LIME on one image, generating a thousand perturbed samples, using any of the methods presented in Subsection 4.1 takes around one minute. While using LIME-G it takes 6. Both of them ran using a Intel Xeon Gold 6240 CPU with 200GB of RAM and a Quadro RTX 8000 GPU with 48GB of video memory. We mentione both, CPU and GPU, given that our implementation of LIME-G executes a part using tensorflow (in the CPUs) and another one using pytorch (in the GPU).

Agarwal and Nguyen [2020] compare LIME and LIME-G and claim that LIME-G consistently yielded explanations that are more accurate and robust to hyperparameter changes. They were, like us, applying LIME on ImageNet examples. They reported that the top 1 predicted class of approximately 20.5% of the LIME perturbation samples was from only three classes: jigsaw puzzle, maze and hen-of-the-wood. Nevertheless, for LIME, they used a shade of gray as replacement color. That, combined with the rectangular shaped superpixels, makes the perturbed samples to look like the original image with a series of gray patches, which indeed resembles a jigsaw puzzle with some missing pieces. In other words, using SLIC's superpixels and gray as replacement color generates perturbed samples that are biased towards those three classes of ImageNet. This could probably gave an advantage to LIME-G over LIME in their experiments. For these reasons, our implementations of LIME and LIME-G are not a replication of theirs, and therefore, they are not fully comparable.

Algorithm 7: LIME-G for images algorithm

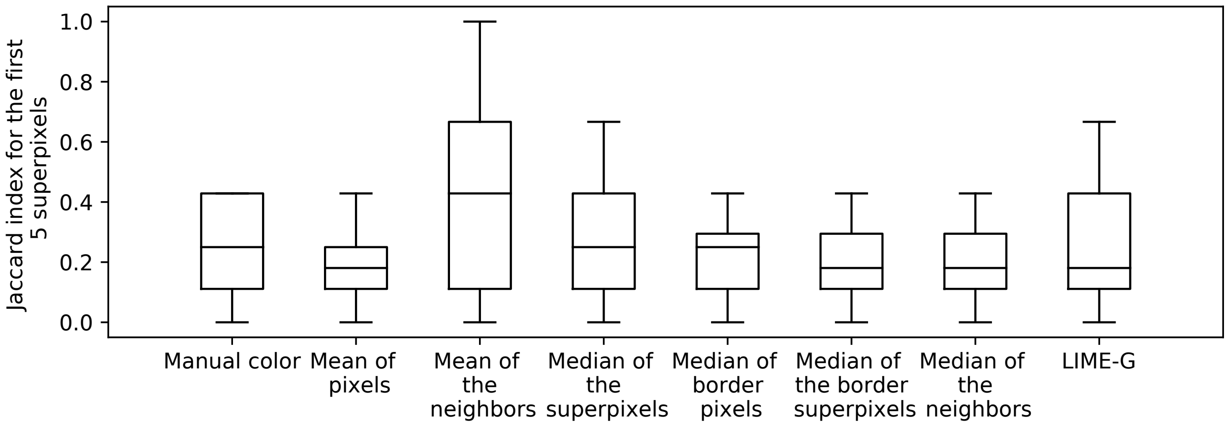
```
Input: an image $\xi$; a classifier $f$; a generative inpainting model $g$, in our case DeepGIN; the number
       $n$ of perturbated images to generate.
Output: the interpretable coefficients $\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_d$.
1  $S \leftarrow$ quickshift($\xi$)
2  $d \leftarrow$ the number of unique values among the entries of $S$
3  for $i \in \{1, 2, \ldots, n\}$ do
4      $z_i \leftarrow$ vector of $d$ i.i.d. random samples from a Bernoulli distribution with parameter 1/2
5      $M_{k,l} \leftarrow (z_i)_{S_{k,l}}$
6      $x_i \leftarrow g(\xi, M)$
7      $y_i \leftarrow f(x_i)$
8      $\pi_i = e^{\frac{-d_{\cos}(1,z_i)^2}{2v^2}}$
9  end
10 $\hat{\beta} \leftarrow$ Ridge regression(features $= z$, responses $= y$, weights $= \pi$)
```

We formalize our implementation of LIME-G in Algorithm 7. The steps are almost the same as in LIME, with some differences: (a) it doesn't generate a replacement image ξ^*; (b) in line 5, the binary mask M is created; (c) subsequently, in line 6, the generative inpainting model (in our case, DeepGIN) takes the image ξ and the mask M as inputs to create the perturbed sample x_i. The rest of the process follows the same logic that was described before for LIME (Section 3).

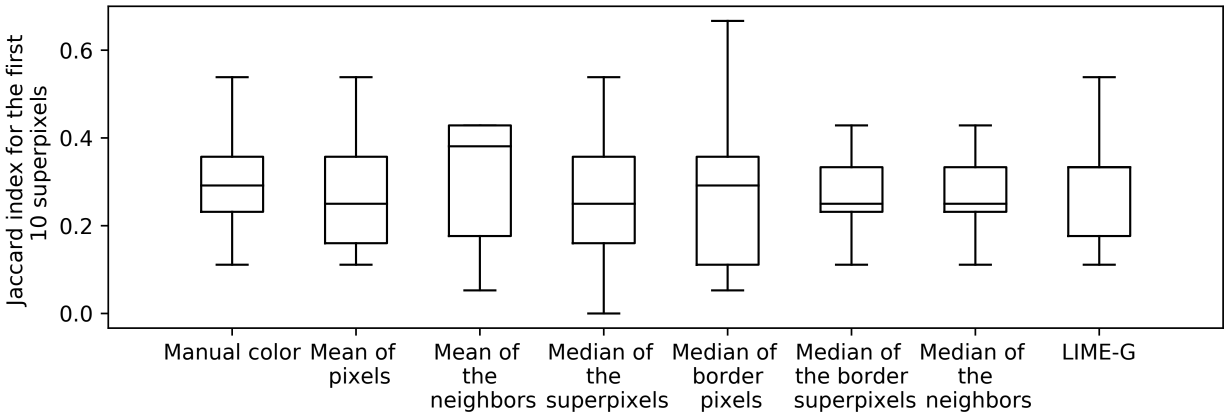
When applying LIME-G to the 12 images where default LIME fails, it provides good explanations for all of them (Figures 20 to 31). Even though it outperforms default LIME and all the replacement methods in the previous subsection for these 12 images, once more, the question that arises is if LIME-G is better than the others in a larger scale, *i.e.*, in a large sample of images. We address this in Section 5 and 6.

5 Similarity among results

Although we can visually inspect the similarities between LIME results for different replacement methods, in this section, we take a quantitative approach to measure how similar the results are using the Jaccard index. We compare each replacement methods and LIME-G, described in previous sections, against default LIME. Subsequently, we focus on the Jaccard index for the results of default LIME, LIME using *median among the mean of the superpixel's neighbors replacement* and LIME-G in a larger set of images: 500 images from the validation set of ImageNet.



(a)



(b)

Figure 13: Jaccard index comparing the top 5 (a) and 10 (b) most important superpixels for default LIME against the other perturbed samples generation methods using 12 images that represent a challenge for default LIME, and taking into consideration only the most probable class according to the classifier.

The Jaccard Index is a similarity metric for finite sets. It is obtained by dividing the size of the intersection by the size of the union of the sets. Formally,

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}.$$

A Jaccard index of 1 indicates that the two sets are equal, hence, maximum similarity. Whereas 0 represents total dissimilarity.

Since we intend to measure the similarity between default LIME results and the different replacement methods, our sets A and B contain the index of the top m most important superpixels according to each method. For instance, let us consider that we are interested only in the top 3 ($m = 3$) most important

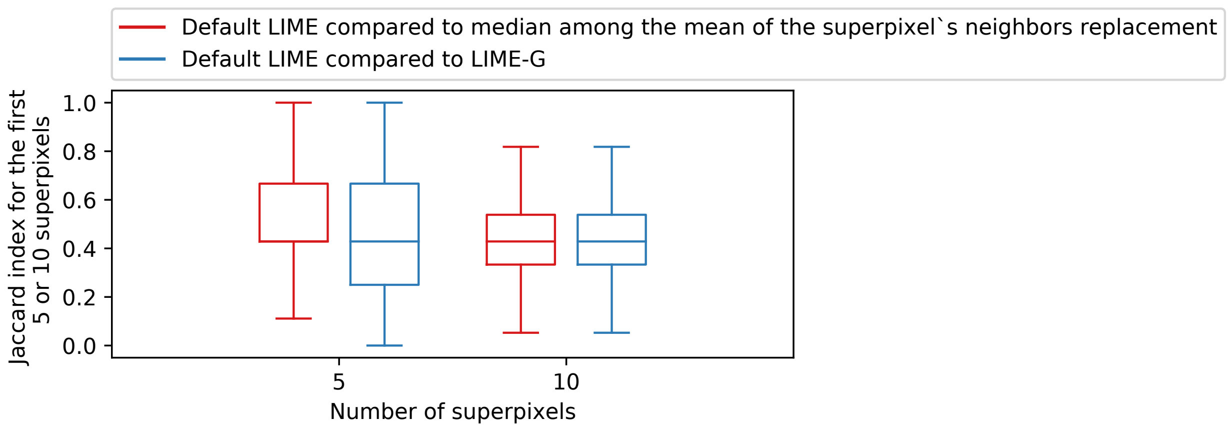


Figure 14: Jaccard index comparing the top 5 and 10 most important superpixels between default LIME, LIME using *median among the mean of the superpixel's neighbors* replacement, and LIME-G. Using the results of the three types of LIME in a sample of 500 images of the validation set of ImageNet.

superpixels. Suppose that default LIME results state that, for a given image, the top 3 corresponds to superpixels 1, 16, and 40. While using the median among the mean of the superpixel's neighbors replacement, the results indicate that the most important superpixels are 2, 15, and 40. The intersection of those two sets contains 1 element (superpixel 40). The union of the sets contains 5 elements (superpixels 1, 2, 15, 16, and 40). Therefore the Jaccard index would be $1/5 = 0.2$.

Note that given a universe U of n elements, and $A, B \subseteq U$ with $|A| = |B| = m$. Where the elements of A are fixed and those of B are randomly sampled from U. The probability of picking k elements from A and $m - k$ elements from $U \setminus A$ is

$$\frac{\binom{m}{k}\binom{n-m}{m-k}}{\binom{n}{m}}.$$

Then the expected value of the Jaccard index for A and B is

$$\mathbb{E}(J(A,B)) = \sum_{k=0}^{m} \frac{k}{2m-k} \frac{\binom{m}{k}\binom{n-m}{m-k}}{\binom{n}{m}}.$$

On the images on which we applied default LIME and its variations, the average number of superpixels produced by the *quickshift* algorithm is 66. There we can assume $n = 66$. If we consider the top 5 (respectively 10) superpixels, $m = 5$ (respectively $m = 10$). The expected value of the Jaccard index between A and B is equal to 0.04 (respectively 0.09). For instance, we expect the Jaccard index between the top 5 superpixels of default LIME and a random guess of 5 superpixels to be 0.04. We will see that the Jaccard similarities observed between our methods are several times higher than what a random guess would produce.

In Figure 13, we observe the Jaccard index between the top 5 (and 10) superpixels of default LIME and the other perturbed sample generation methods. This for the 12 images that result challenging for default LIME. In general, the one that shows more similarities with default LIME is *mean of the neighbors replacement*. Nevertheless, it also shows the greater variability in the top 5 scenario. In regards to which method provides the more dissimilar results, *median of the mean of the superpixel's neighbors*, and the two replacements related to the border present the smallest medians in both cases. However, given that we are using only 12 images, it is difficult to establish if there are important differences among them.

According to the visual evaluation of the explanations, the *median among the mean of the superpixel's neighbors replacement* and LIME-G gave the best results, we applied these methods, and default LIME, to a sample of 500 images from the validation set of ImageNet from the 2012 challenge. This sample contains images that are well-classified by inceptionV3. Using those results we computed the Jaccard index for the top 5 and 10 superpixels (Figure 14). In both cases, the metric is slightly higher than 0.4 for *median among* *the mean of the superpixel's neighbors* and LIME-G. Generally, the Jaccard similarities are 10 times higher than what a random guess would produce, in the case of top 5 superpixels, and around 7 times higher in the case of top 10 superpixels. Therefore, there is a substantial overlap. Nevertheless, we can see that the variability drags the whiskers until the extremes, 0 and 1.

6 Evaluation of the explanations

Previously, we observed the (dis)similarities between default LIME results and other methods. However, such discrepancies do not state that these methods produce better explanations than default LIME. In this section, we define metrics to evaluate the explanations.

6.1 Evaluation methods in the literature

Let us first consider the evaluation metrics that have been used for LIME and other interpretability methods. We must highlight that there are no established ground-truth datasets to evaluate the correctness of explanations, prior research assessed correctness via proxy metrics. In this subsection, we group the evaluation methods by the authors who used them.

Ribeiro et al. [2016]. They evaluated the utility of LIME by performing experiments with and without human users.

For the experiments not involving humans, they intended to address the faithfulness of the explanations to the model, the trustworthiness of the explanations, and if they could be useful to evaluate the model. They used datasets where the instances were texts; the features, bags of words; and the classifiers belong to the binary classification setting. They considered three other explanation methods besides LIME:

- Parzen [Baehrens et al., 2010]: This method approximates the black box classifier globally with Parzen windows, and explains individual predictions by taking the gradient of the prediction probability function. For the experiments, they took the 10 features with the highest absolute gradients as explanations. This method, as LIME, uses a linear model.
- Greedy: This technique consists in removing one by one the features that contribute the most to the predicted class until the prediction changes or until we reach the maximum of 10 features.
- Random: It randomly picks 10 features as explanations. This corresponds to the *control method.*

To select the instances that would be used in the experiments, they considered two picking processes: random pick, which just applies random selection, and submodular pick. To apply the second one, we must rank features according to their importance in a picking process. However, in their example, they were considering a bag of words as features. In our case, we can not compare the first superpixel of an image, to the first superpixel of another image. Therefore, submodular pick might not be an option for evaluating our results.

a. The first experiment aimed to answer the question "are explanations faithful to the model?". To measure faithfulness, they train two classifiers that are inherently interpretable (sparse logistic regression and decision trees) in such a way that they would use maximum 10 features. These 10 features are considered the *true explanations.* They generate the explanations with the 4 methods listed before, and then they calculate the percent of the explanations that belong to the *true explanations.*

b. The second experiment measured the trust in the predictions. They defined *untrustworthy features* and *untrustworthy instances* in the following way. They randomly select 25% of the features and we assumed that they were untrustworthy. Using the classifier, they defined an instance as untrustworthy if the predicted class changes when the untrustworthy features are removed from the instance. These untrustworthy instances are considered the ground truth. It is untrustworthy because those features were chosen randomly, hence most of the time they are not the most important features, removing them should not cause a change in the predicted class. Let us remember that the features were a bag of words, so removing a feature means setting 0 for that feature in the bag of words. For LIME we define that an instance is untrustworthy if the predicted class, obtained using the surrogate linear model, changes when all untrustworthy features that appear in the explanations are removed. The same applies to Parzen. For greedy and random, the instance is defined as untrustworthy if any untrustworthy features are present in the explanation. Given that the untrustworthy features were selected randomly at the beginning, the process was repeated 100 times. They measured the F1-score, recall, and precision on the trustworthy instances for each explanation method (compared to the assumed ground truth). In their example, identifying a change in class using the surrogate model of LIME is simple because they were in the case of binary classification. That is, they would just measure if the probability given by the model is smaller or greater than 0.5. However, in our case, the result of our surrogate model intends to predict the probability of belonging to the class in which the original image was classified. If the untrustworthy features that appear in the explanations are removed, the increase or decrease in the predicted probability will not indicate a change in the class. Hence, we could opt to use with LIME, the same criteria as in the greedy and random procedure to determine the untrustworthy instances.

c. The third experiment's objective was to determine if the explanations can be used for model selection. In this experiment they simulate a human selecting between two models. They added 10 artificial "noisy" features, which are considered untrustworthy. They trained pairs of classifiers using random forests with 30 trees until their validation accuracy is within 0.1% of each other, but their test accuracy differs by at least 5%. Hence, it is not possible to identify the better classifier (the one with higher test accuracy) from the accuracy of the validation data. Then they calculate the number of untrustworthy instances among the instances in the test set (with the same procedure described in the second experiment). Among the two classifiers, the one with fewer untrustworthy instances is selected. They compare this choice to the classifier with higher held-out test set accuracy to determine if using the explanations the best model was selected. This process was run 800 times, and they considered the percent of the times that the good classifier was selected.

For the cases without humans, they developed three experiments:

a. The first one aimed to evaluate if users can choose which classifier generalizes better based on the explanations. Two classifiers were trained. One in a "cleaned" dataset where the features that do not generalize have been manually removed, and the other in the original dataset. The one trained with the clean dataset is considered the best. The raw data and the explanations (using LIME and greedy methods) were presented to 100 human subjects. They measured the percent of times that the good classifier was selected thanks to the explanations.

b. The second experiment measured if users can perform feature engineering based on the explanations. Using the worst classifier from the previous experiment, they extracted the explanations. Users were asked to identify which features from the explanations should be removed from subsequent training. That is, the subject marked words for deletion after observing 10 instances with 10 features in the explanation. In the first round, 10 subjects suggested changes, and those changes were applied, which creates 10 new classifiers. The explanations for each classifier were presented to 5 users in the second round, which results in 50 new classifiers. The third round involved other 5 subjects, which created 250 classifiers. The accuracy of these classifiers was measured. In such a way, we can identify if the explanations help to improve the classifier.

c. The third and final experiment determined if users are able to identify and describe classifier irregularities by looking at explanations. To measure that, they intentionally trained a classifier that performs well but for the wrong reasons. They trained a logistic regression to identify if an image is a wolf or a husky using 20 images, such that all pictures of wolves had snow in the background, while pictures of huskies did not. A set of 10 pictures with their respective predictions (without explanations) was presented to the subject. Among the 10 pictures, one wolf was not in a snowy background (and thus the prediction was Husky) and one husky was (and was thus predicted as Wolf). The other 8 examples are classified correctly. The subjects had to answer if they trusted the algorithm, why, and how did they think the algorithm was able to distinguish between wolves and huskies. Then the same images were presented to the subjects, but this time adding the explanations, and the same questions were asked. Given that the exercise requires some notion of spurious correlations and generalization, in this last experiment, the subjects were graduate students who have taken at least one graduate machine learning course.

Fong and Vedaldi [2017]. They propose an explainability method similar to integrated gradients. This method belongs to the family of saliency maps and produces a heatmap that highlights the most important pixels. To evaluate their method they performed various experiments.

a. First, they implemented several saliency maps methods in some images, and visually compare them to determine which method could pinpoint the key elements of the images without highlighting non-essential evidence.

b. They also use a more quantitative criterion. Given that saliency map methods produce a mask that perturbs the image, they used these perturbed images to quantify the drop in the predicted probability of the class with respect to the original image. They compared the different techniques looking for the one that causes the maximal drop.

c. The authors knew that the masks produced by the method they propose could be changing the predicted probability by creating adversarial examples instead of proper explanations. Therefore, to test that the masks were generalizable and robust, they simplified the masks by blurring them and then slicing them into binary masks. Subsequently, they used the simplified masks, which no longer provoke adversarial examples, on the images, and measured the drop in the predicted probabilities.

d. Adversarial examples were also at the center of another experiment. In this case, they generated adversarial examples and applied different saliency maps methods to them, and to clean images. Then, they trained a classifier that uses the produced masks as input and predicts if the image is an adversarial example or not.

e. Another experiment consisted of analyzing the ability of the method to correctly identify a minimal region that contains the most important elements in the image. For this, they fitted the tightest bounding box around the resulting heatmap. Then, they blurred the image in the box and compute the predicted probability. They fixed the desired decrease percent in the probability and evaluated which saliency map method produced the smaller bounding boxes.

f. Finally, they intended to test how discriminative the explanations are by determining if the maximum point of the heatmap lies on the most important element in the image. They defined a bounding box that encloses the key part of the image as ground truth. Later, the maximum points of the heatmaps produced by different saliency methods were identified using bounding boxes (based on different thresholding methods). They calculated the precision of these boxes compared to the ground truth.

Lundberg and Lee [2017]. They propose another explainability method, SHAP (Shapley Additive Explanations), and some variations of it (KernelSHAP and DeepSHAP). These methods are modifications of Shapley sampling values. To evaluate their method, they compare it against other methods in terms of computational efficiency, consistency with human intuition, and its ability to explain class differences.

a. For the first experiment, they compared the computational efficiency and accuracy of Shapley sampling values, Kernel SHAP, and LIME on both dense and sparse decision tree models. The accuracy was determine with respect to a ground truth Shapley value.

b. The second experiment involved human intervention. They compared the explanations from LIME, DeepLIFT, and SHAP with user explanations of simple models. They assumed that model explanations should be consistent with explanations from humans who understand that model.

c. Finally, using the MNIST dataset, they compared the change in log odds of belonging to a class when using the mask produced by LIME, DeepLIFT, and SHAP.

Agarwal and Nguyen [2020]. As stated before, they were the first to combine generative inpainting and LIME, resulting in LIME-G (explained in section 4.2). Besides LIME, they modify other two explainability methods based on perturbed samples (Sliding-Patch, and Meaningful-Perturbation). As in LIME-G, when some pixels need to be *turned off* in either of these algorithms, they use a generative inpainting model to fill them. They compare the three algorithms against their modified versions, which they call the generative methods.

They recognize that there are currently no established ground-truth datasets to evaluate the correctness of explanations. Nevertheless, to evaluate if the explanations provided that generative methods are more accurate they used three proxy methods.

a. The first one is the object location task. For this method, the explanations are visualized as a heatmap. In the case of LIME is means to color the superpixels based on their coefficients. Then multiple bounding boxes are created on the heatmap by thresholding it at different values of $t = \alpha\mu_{max}$, where μ_{max} is the maximum intensity in the heatmap and $\alpha \in \{i \cdot 0.05 \quad : \quad 0 \leq i \leq 20\}$. For each α, they computed the intersection over union score between a derived bounding box and the ground-truth bounding box. For each explanation method, they chose the best α^* that yielded the lowest error.

b. The second proxy score was the deletion metric, which measures the area under the curve of the target-class probability as they gradually turn off input pixels of the highest importance in descending order. Lower deletion scores indicate more accuracy. The authors did not turn off pixel by pixel, but groups of them.

c. The third one was the saliency metric. In the same manner as the object location task, several bounding boxes are generated. The section of the image inside each bounding box is given to the classifier to obtain the probability of belonging to the target class. Then the saliency metric is computed using

$$\log(\max(a, 0.05) - \log(s(x_p))),$$

where a is the ratio of the area of the bounding box over the image size and $s(x_p)$ is the classification probability. The smallest bounding box which yielded the least salient metric is selected. A lower saliency score indicates better accuracy.

d. The authors also test whether generative methods are more robust to hyperparameter changes than their original counterparts. They compare LIME and LIME-G, by running both of them 5 times in each image, using different seeds. Then they compared the similarity of the results using the Structural Similarity Index (SSIM), the Pearson correlation of the histograms of oriented gradients (HOGs), and the Spearman rank correlation. The results indicate that LIME explanations change by changing the random seed. Whereas LIME-G is consistently more robust than LIME. Nevertheless, they also realize that LIME perturbed samples usually were classified in the classes jigsaw puzzle and maze (as explained in Section 4.2). This occurs because the type of segmentation they used for the superpixel partitions the image in square-like pieces that, when turned gray, can easily resemble a puzzle. This phenomenon was not observed in the perturbed samples of LIME-G.

As it can be observed in this literature review, a common factor for all authors is to measure the change in the probability of belonging to the predicted class and establish metrics around it. We follow this approach in the next two subsections to evaluate the quality of the explanations.

6.2 Delta among the predictions

Let us consider one of the 12 images that are changing for classic LIME. This image shows a camel. InceptionV3 indicates that there is a probability of 0.61 of the image to belong to the class Arabian Camel. If we somehow remove the superpixels that contain the silhouette of the camel and pass this new image through the classifier, we expect the probability of belonging to that class to drop. Hence, if the replacement method correctly identifies the most important superpixels, and we turn these superpixels off, we expect a drop in the probability of belonging to the class greater than by using other methods that do not identify the most important superpixels. Therefore, a greater decrease would indicate better explanations.

In Figure 15, we replaced the top 5 most superpixels (according to the results of each replacement method) with their counterparts in their respective replacement image. We passed each of these images through InceptionV3 and obtain the probability of belonging to the Arabian Camel class. For example, using mean replacement (second image), *i.e.*, default LIME settings, the probability drops from 0.61 to 0.15. In other words, it drops by 0.46. In this example, the *mean of the superpixel's neighbors* is the replacement type that causes the biggest decrease, 0.6. Given that, in the original image, the probability of belonging to the Arabian Camel class was 0.61, this decrease means that, in the new image, the probability is 0.01.

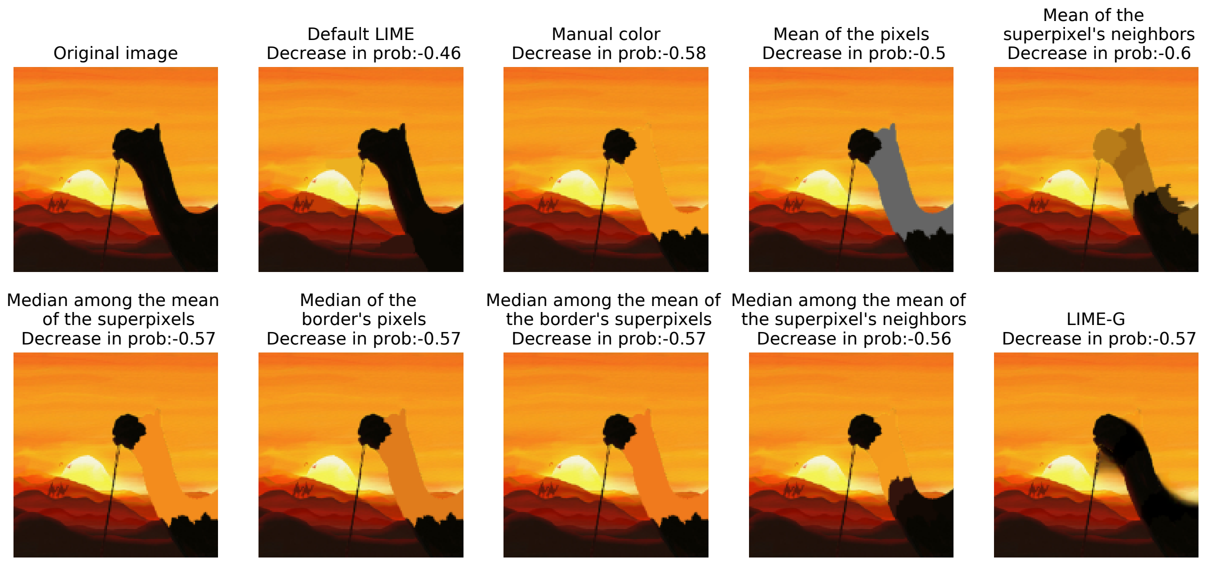


Figure 15: Images where the 5 most superpixels (according to the results of each method) were *turned off*. We pass these images through the classifier and observe the drop in the probability of belonging to the Arabian Camel class with respect to the original image.

We measured the decrease in the probability as well as the percent of change related to the decrease for the images where default LIME fails.

In Table 2, we can see the differences in probability for those 12 images. Notice that, in some cases, the probability augments. In some cases, the decrease is minimal and very similar among all types of replacements, like in the tarantula example. This could be related to the fact that we are extracting the top 5 superpixels, but in the tarantula case, ideally, we should extract just the top 1, since the tarantula is enclosed by one superpixel only.

In relative values (Figure 16), considering the median of the percent of change in the probability, *manual color replacement* followed by *median among the mean of the superpixels in the border* and LIME-G caused the greater drop in the probability, among the 12 images where default LIME fails. Given that many of the methods have some cases where the decrease is of a 100%, they cause the probability to be zero or close to zero. But, most importantly, all the methods caused a greater decrease in the probability than default LIME, and they exhibit less variance in such drop.

Now we consider the sample of 500 well-classified images from the validation set of Imagenet 2012 and examine the drop only for default LIME, *median among the mean of the superpixel's neighbors*, and LIME-G. Both the median of the drop in the predicted probability and its variability are similar between the three techniques (Figure 17).

Table 2: Decrease in the probability of belonging to the class of the original image, for default LIME and the different perturbed sample generation methods, applied in the group of 12 images that result challenging for default LIME. Bold numbers indicate the greatest decrease for each image. Lower numbers indicate better explanations.

Image	Default LIME	Manual color	Mean of pixels	Mean of the neighbors	Median of the superpixels	Median of border pixels	Median of the border superpixels	Median of the neighbors	LIME-G
laptop	-0.03	-0.29	-0.33	-0.25	-0.28	-0.31	-0.32	0.13	**-0.33**
band aid	-0.24	-0.22	-0.25	**-0.25**	-0.24	-0.21	-0.19	-0.24	-0.24
otter	-0.35	-0.16	**-0.36**	-0.33	-0.15	-0.2	-0.34	-0.34	-0.36
warplane	-0.13	-0.56	-0.59	-0.59	-0.58	-0.58	-0.59	**-0.59**	-0.58
oystercatcher	-0.26	**-0.26**	-0.26	-0.26	-0.26	-0.26	-0.26	-0.26	-0.26
arabian camel	-0.46	-0.58	-0.5	**-0.6**	-0.57	-0.57	-0.57	-0.56	-0.57
tarantula	-0.06	-0.07	-0.07	-0.07	-0.07	-0.07	-0.07	-0.07	**-0.07**
magpie	-0.15	-0.2	-0.2	-0.11	-0.2	-0.2	**-0.2**	-0.2	-0.2
sliding door	-0.23	**-0.46**	-0.19	-0.42	-0.27	-0.14	-0.3	0.05	-0.25
daisy	-0.02	0.06	0.04	-0.01	**-0.04**	-0.04	0.0	0.01	0.03
hip	-0.64	-0.92	-0.91	-0.92	-0.92	**-0.92**	-0.92	-0.81	-0.39
jean	-0.12	-0.13	0.16	-0.1	-0.12	-0.13	**-0.13**	-0.12	-0.13

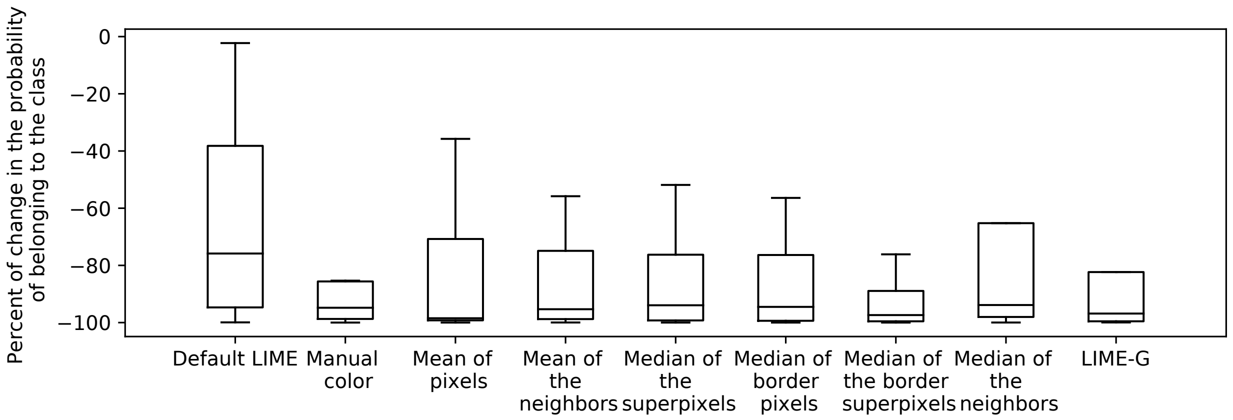


Figure 16: Percent of change of the decrease in the probability of belonging to the class of the original image for default LIME and the other perturbed samples generation methods, using 12 images that represent a challenge for default LIME and taking into consideration only the most probable class according to the classifier.

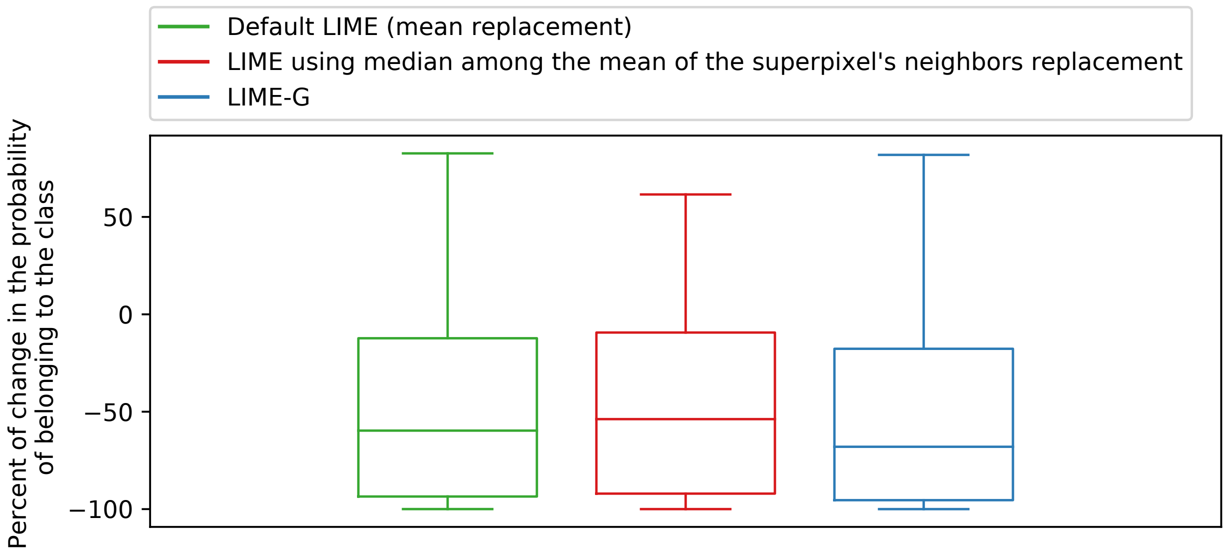


Figure 17: Percent of change of the decrease in the probability of belonging to the class of the original image, default LIME, median among the mean of the superpixel's neighbors replacement, and LIME-G. Using the results of the three types of LIME in a sample of 500 images of the validation set of Imagenet.

6.3 Deletion metric

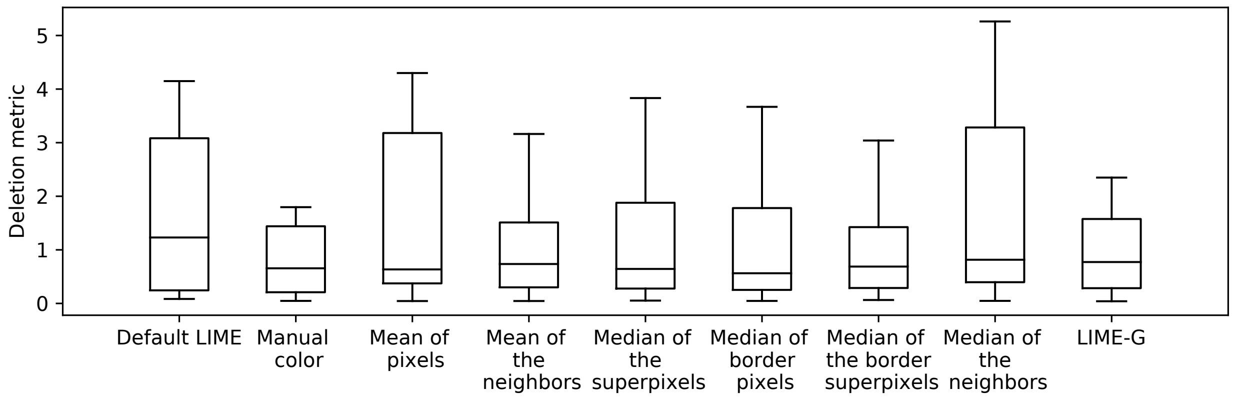


Figure 18: Deletion metric for default LIME and the other perturbed samples generation methods, using 12 images that represent a challenge for default LIME and taking into consideration only the most probable class according to the classifier. Lower scores are better.

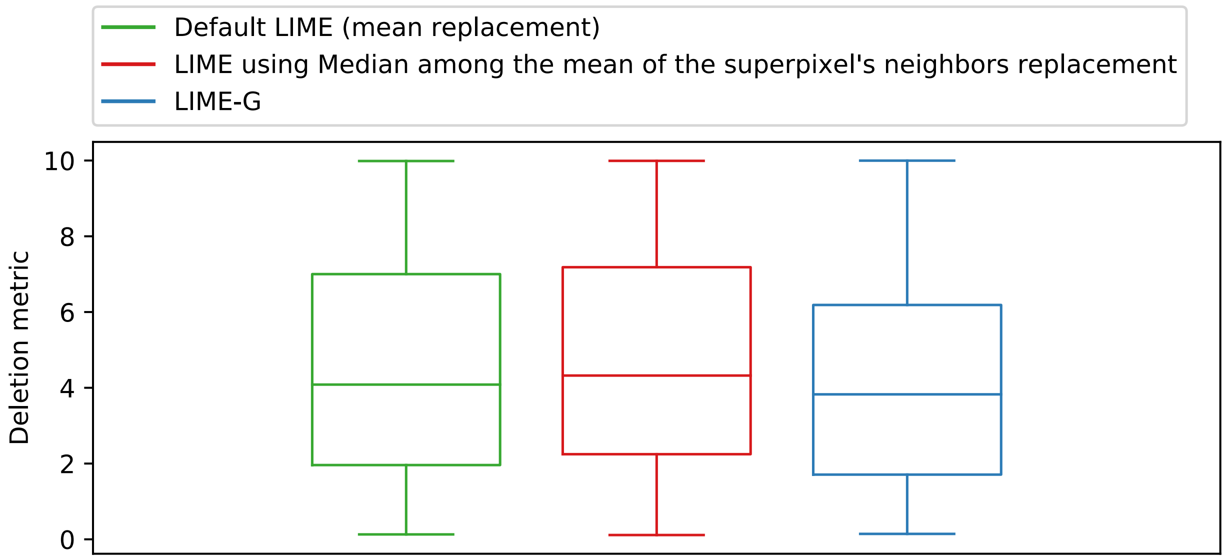


Figure 19: Deletion metric for default LIME, LIME using median among the mean of the superpixel's neighbors replacement, and LIME-G. In a sample of 500 images of the validation set of Imagenet. Lower scores are better.

In Subsection 6.1, we mentioned the deletion metric. Where it was defined as the area under the curve of the probability of belonging to the predicted class as we gradually turn off input pixels of the highest importance in descending order. Lower deletion scores indicate more accuracy. Usually, this process is not done pixel by pixel but on groups of them. In our case, such groups of pixels are determined by the superpixel. That is, we first get the probability of belonging to the class for the original image. Such probability determines de first point that we "draw". Subsequently, we *turn off* the most important superpixel according to the interpretability method (either default LIME or any of the variations) and pass this new image through the classifier f to get the new probability of belonging to the class. This second probability determines de second point. Then, we *turn off* the 2 most important superpixels and get the probability (which defines the third point), and so on until a certain limit of superpixels are *turned off.* We fixed such a limit of superpixels in 10. Then we draw a line to connect the points and get the area under such curve.

We measured the deletion metric for default LIME and its variations using the other perturbed samples generation methods on the 12 images where default LIME fails (Figure 18). Default LIME had the worse median deletion score, while all the other ones hold a similar median score, and *median among the mean of superpixel's neighbors* presented the greatest variability.

Subsequently, we considered the sample of 500 well-classified images from the validation set of Imagenet 2012. We got the deletion metric using default LIME, *median among the mean of the superpixel's neighbors*, and LIME-G (Figure 19). LIME-G presents the best median deletion score, followed by default LIME and, at last, *median among the mean of the superpixel's neighbors.* Nevertheless, the differences between them are not large. Their variances are also similar, especially between default LIME and LIME using *median among the mean of the superpixel's neighbors.*

7 Discussion

Several questions and ideas risen through this study can be explored in future analysis. First, LIME-G did not show a significant improvement in the explanations, on the large scale, compared to default LIME. This opposes the results of Agarwal and Nguyen [2020]. We suspect that in their case LIME-G could have been favored mostly due to the segmentation type, which created perturbed samples biased toward classes like "jigsaw puzzle" for default LIME. This exposes the importance of the segmentation algorithm and its parameters to LIME explanations, which could be subject of a future study.

Second, a combination of semantic segmentation and inpainting models could be used to "erase" sections of the image more realistically. Inpainting models have an easier time removing complete objects from the image than sections of them. For instance, consider the lion image shown in Figure 12. When the whole region of the lion is inpainted, we end up with a more realistic image than when sections of the lion are inpainted. In the second case, when only some sections of the lion are inpainted, the lion can even "expand" creating black blurs. Semantic segmentation models have the ability to fragment an image according to the objects on it, defining better superpixels. Given that those would contain entire objects, the inpainting model could produce better results when turning them off.

LIME could also take advantage of background detection models as some of the methods that we explored aim to erase parts of the image using shades of color from the background or borrowing patterns from it. This gives us ground for another perturbed sample generation method that could be implemented.

Additionally, our evaluation metrics were related to measuring the change in the probability of belonging to a class. However, as explained in Section 6.1, we could apply many others to these same experiments. Those could expound how some of the perturbed sample generation techniques can be more advantageous than others. For instance, the object location and saliency metrics used by Agarwal and Nguyen [2020].

8 Conclusion

We have implemented eight variations of LIME on which we change the method to generate the perturbed samples. Seven of those are based on changing the process to creates the replacement image. Our results point the *median among the mean of the superpixel's neighbors* method as the most promising among them. The other technique, LIME-G, leverages inpainting models to produce the perturbed samples.

We tested those methods, first, on a small sample of images proved to be challenging for default LIME. Later we tried some of the methods on a larger sample of 500 images combining both, instances for which default LIME succeeds to provide good explanations and others on which it does not.

We measured the similarities, using the Jaccard index, among the default LIME and the versions with the other perturbed sample generation methods. The median of the Jaccard index indicates that the similarities are several times higher than what a random guess would produce. This signals overlaps in the explanations of the different techniques.

The quality of the explanations was measured using the percent of the decrease in the probability and the deletion metric. These metrics indicate that the new methods improve the explanations in images that are challenging for default LIME. However, in general, there are not notorious differences between default

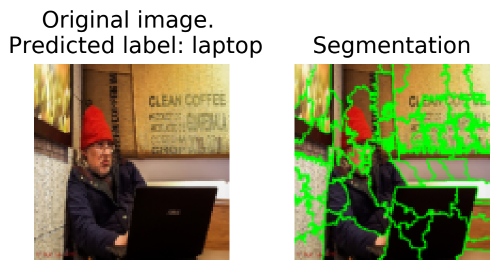
LIME, LIME using *median among the mean of the superpixel's neighbors*, and LIME-G. Due to the higher computational cost of LIME-G, this method would be the least preferred of the three.

9 Appendix

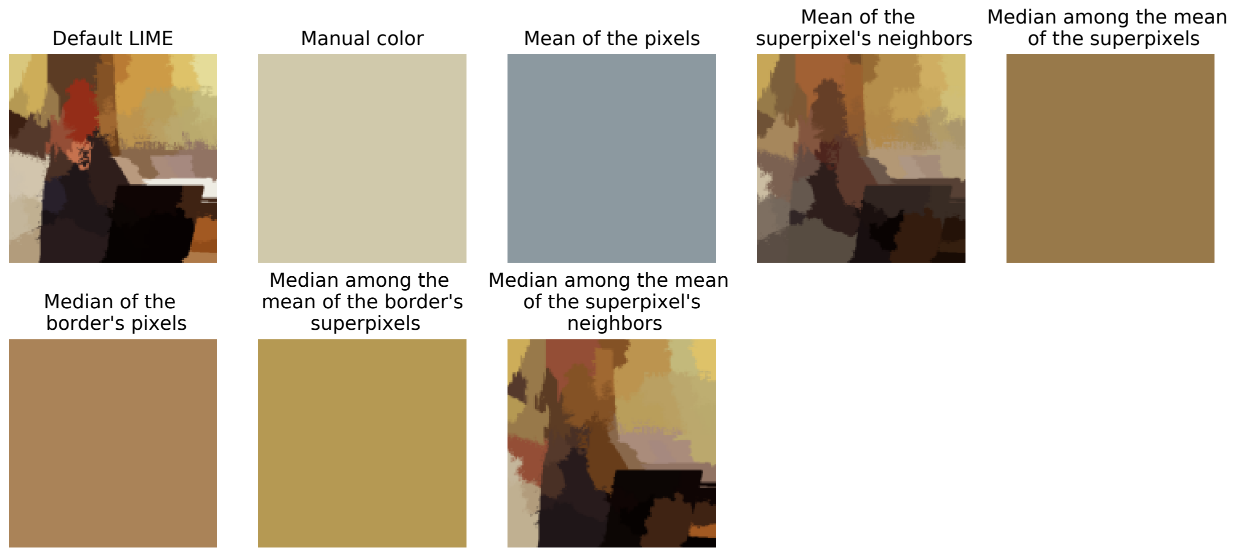
In this appendix, first, we present the results of default LIME and LIME using different perturbed sample generation methods (Subsection 9.1) on 12 images that represent a challenge to default LIME. Each figure shows (a) the original image and its segmentation; (b) the replacement image ξ^* obtained according to each replacement technique, note that LIME-G does not use a replacement image; and (c) the explanations for each method, that is, the top 3 most important superpixels are highlighted in green.

Second, in Subsection 9.2, we present an experiment related to the mainline of work. We varied the parameters of the *quickshift* algorithm to generate greater fragmentation, *i.e.*, produce more superpixels, and observe the changes on the explanations given by default LIME and LIME using *median among the mean of the superpixel's neighbors.*

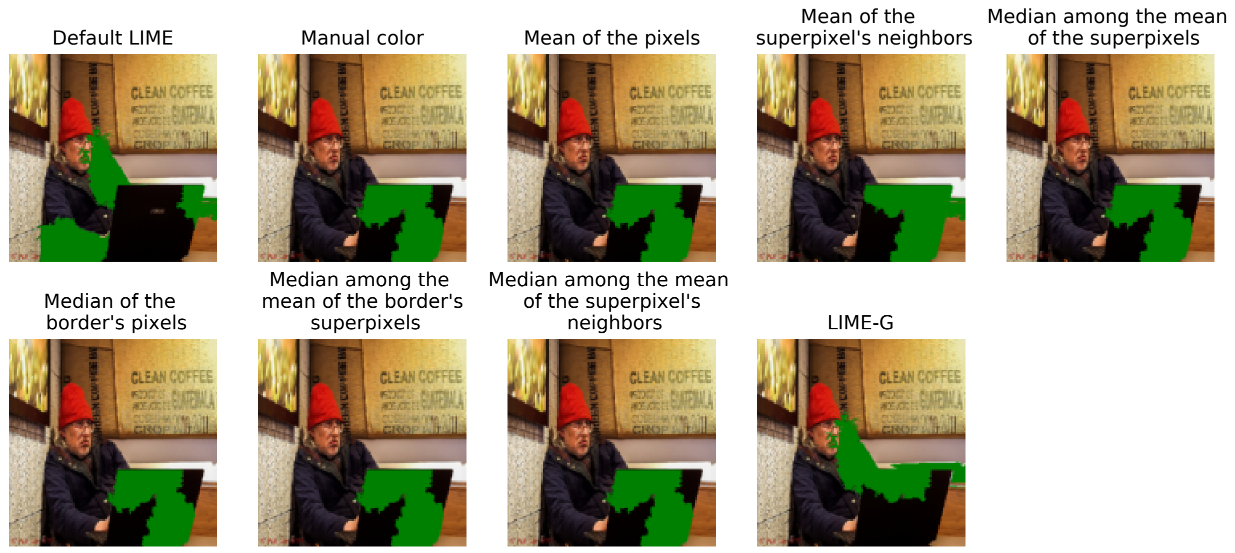
9.1 Results on images that are challenging for default LIME



(a) Original image and segmentation

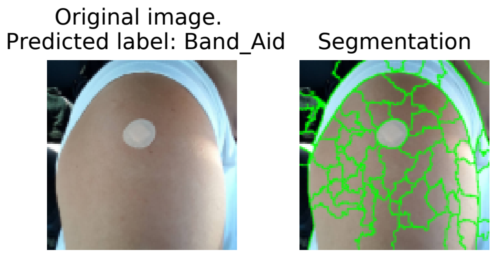

Default LIME
Manual color
Mean of the pixels
Mean of the superpixel's neighbors
Median among the mean of the superpixels
Median of the border's pixels
Median among the mean of the border's superpixels
Median among the mean of the superpixel's neighbors

(b) Replacement images

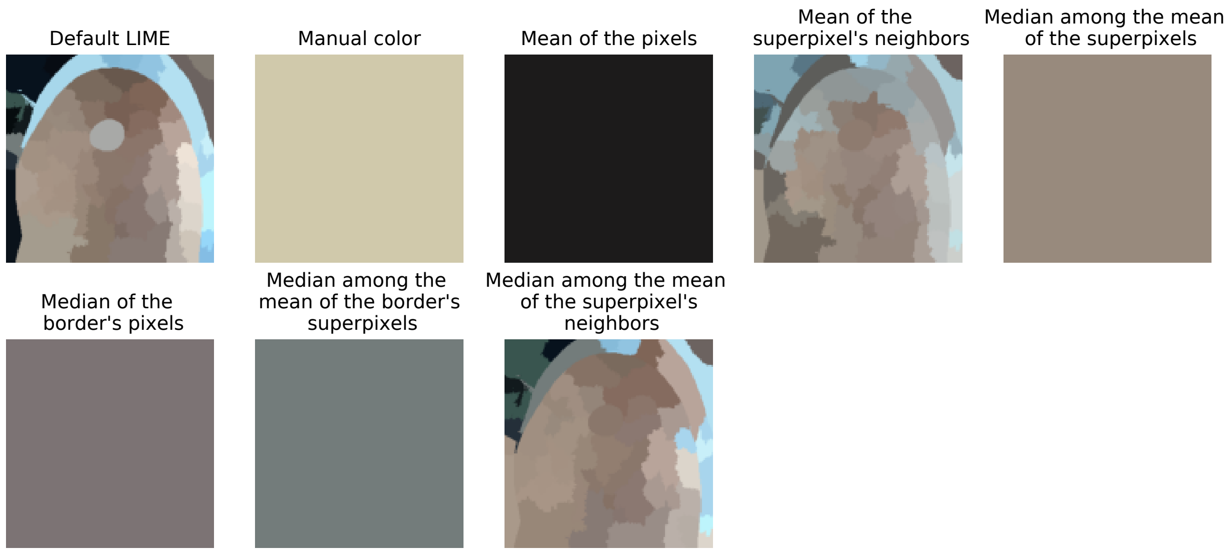

Default LIME
Manual color
Mean of the pixels
Mean of the superpixel's neighbors
Median among the mean of the superpixels
Median of the border's pixels
Median among the mean of the border's superpixels
Median among the mean of the superpixel's neighbors
LIME-G

(c) Explanations

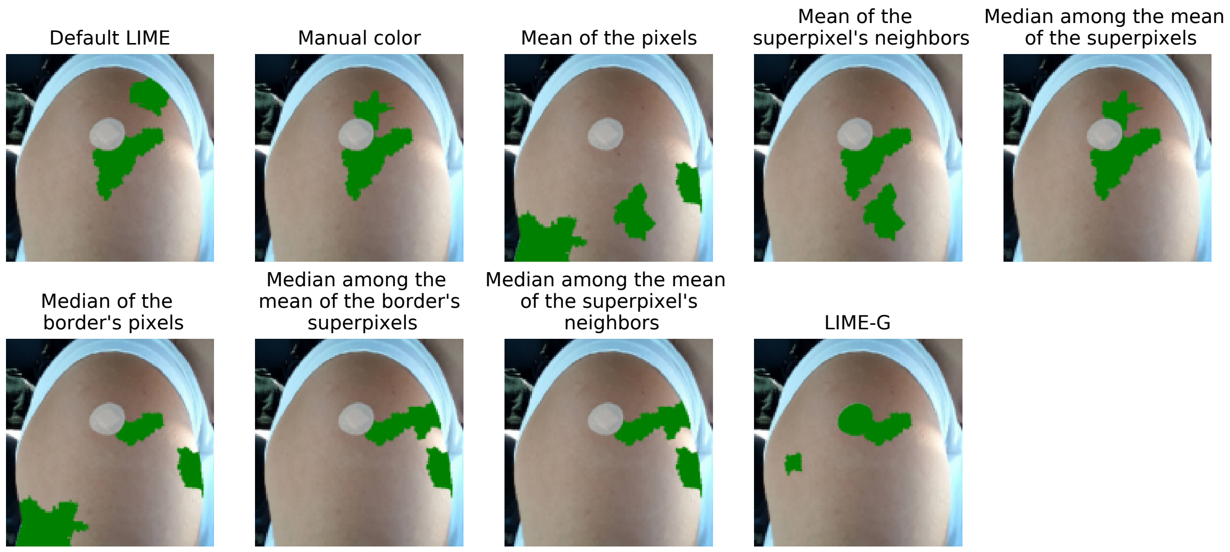
Figure 20: Comparison of LIME using different perturbed samples generation methods of an image where default LIME fails. Original image, superpixel segmentation, and replacement image and explanations (top 3 superpixels) for each perturbed samples generation method.



(a) Original image and segmentation

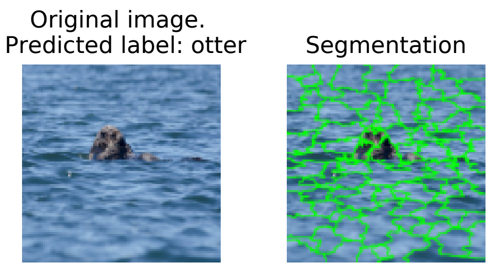

Default LIME
Manual color
Mean of the pixels
Mean of the superpixel's neighbors
Median among the mean of the superpixels
Median of the border's pixels
Median among the mean of the border's superpixels
Median among the mean of the superpixel's neighbors

(b) Replacement images

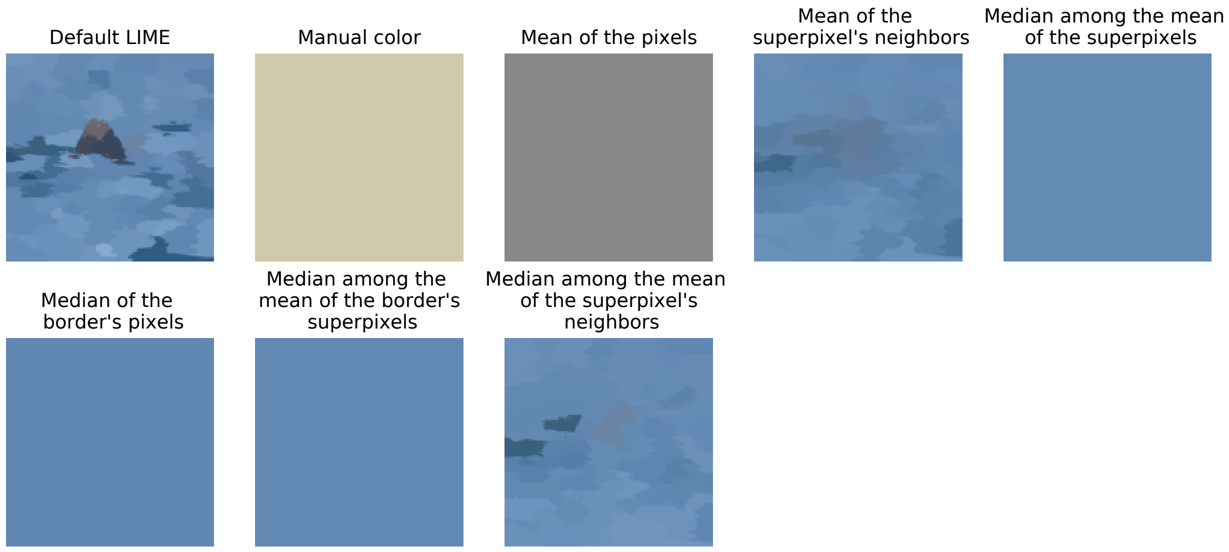

Default LIME
Manual color
Mean of the pixels
Mean of the superpixel's neighbors
Median among the mean of the superpixels
Median of the border's pixels
Median among the mean of the border's superpixels
Median among the mean of the superpixel's neighbors
LIME-G

(c) Explanations

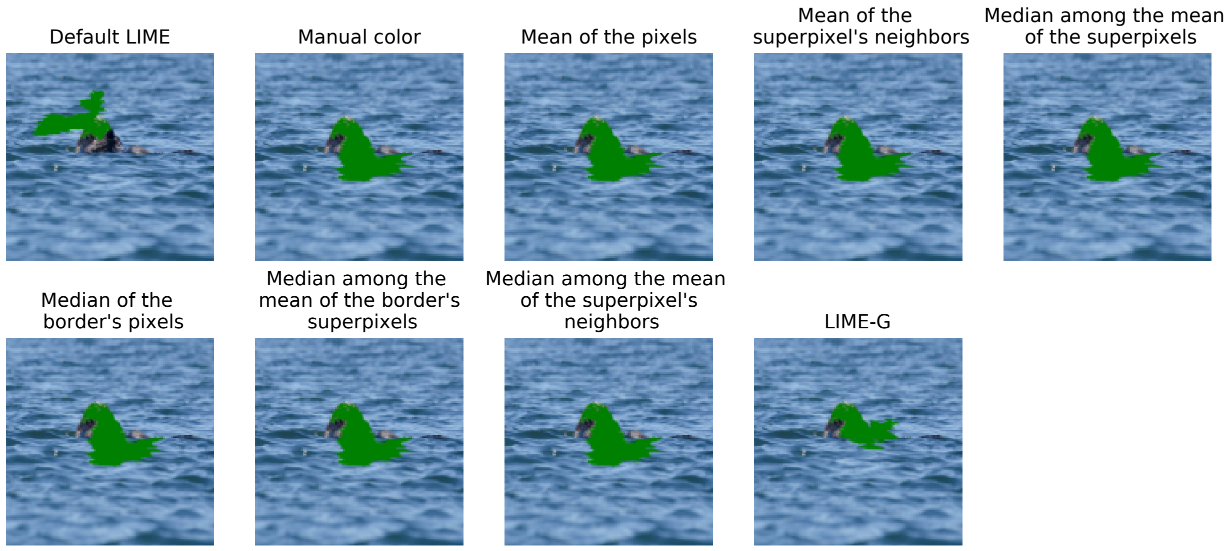
Figure 21: Comparison of LIME using different perturbed samples generation methods of an image where default LIME fails. Original image, superpixel segmentation, and replacement image and explanations (top 3 superpixels) for each perturbed samples generation method.



(a) Original image and segmentation

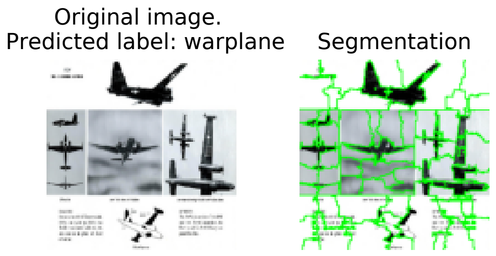

Default LIME
Manual color
Mean of the pixels
Mean of the superpixel's neighbors
Median among the mean of the superpixels
Median of the border's pixels
Median among the mean of the border's superpixels
Median among the mean of the superpixel's neighbors

(b) Replacement images

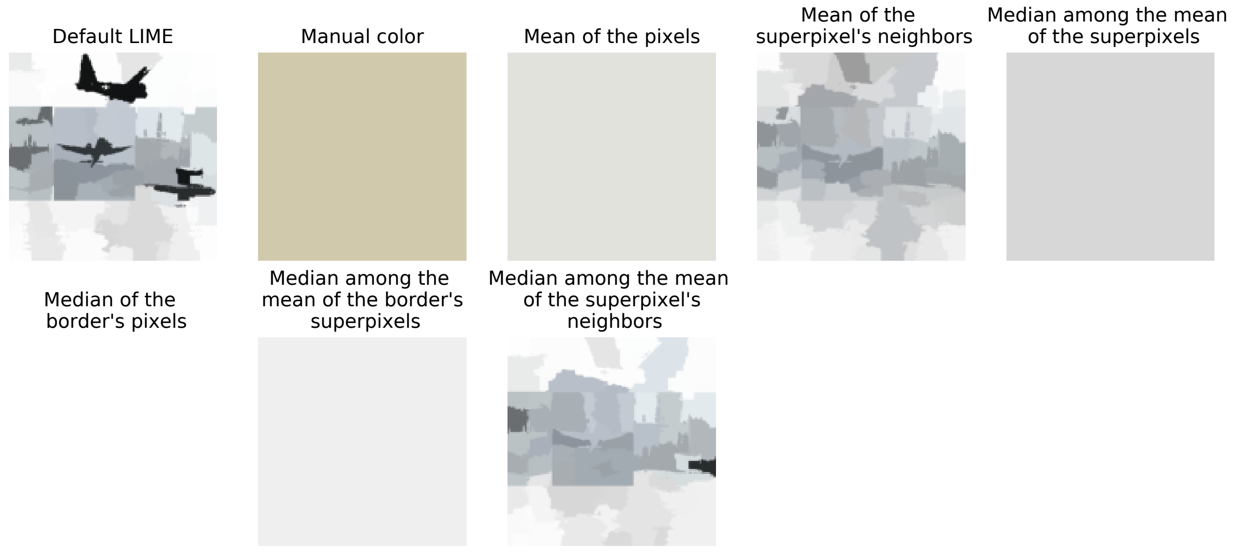

Default LIME
Manual color
Mean of the pixels
Mean of the superpixel's neighbors
Median among the mean of the superpixels
Median of the border's pixels
Median among the mean of the border's superpixels
Median among the mean of the superpixel's neighbors
LIME-G

(c) Explanations

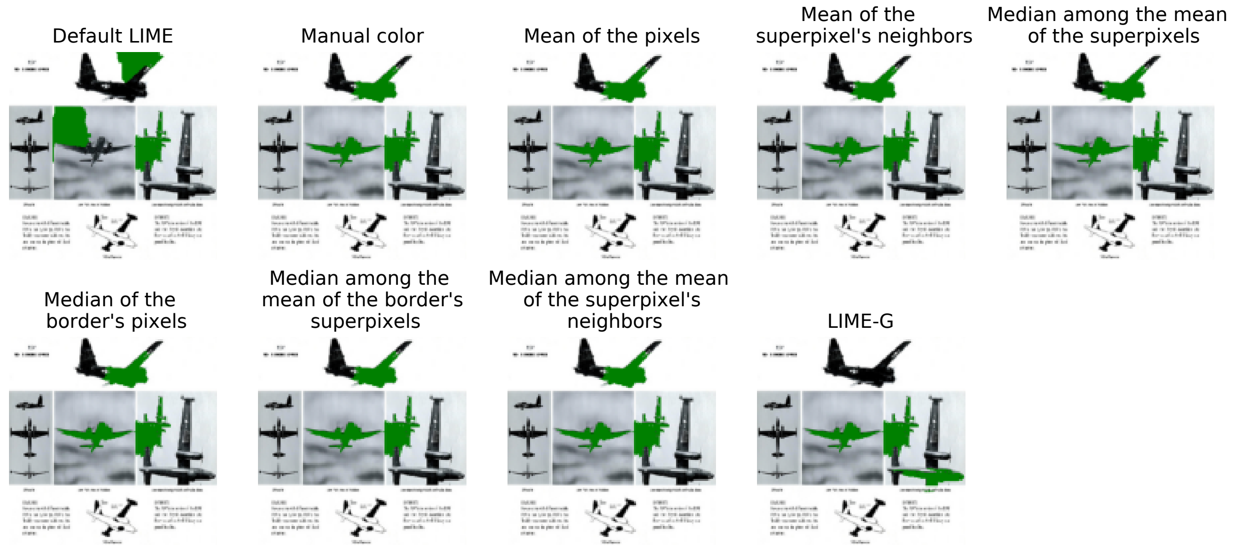
Figure 22: Comparison of LIME using different perturbed samples generation methods of an image where default LIME fails. Original image, superpixel segmentation, and replacement image and explanations (top 3 superpixels)for each perturbed samples generation method.



(a) Original image and segmentation

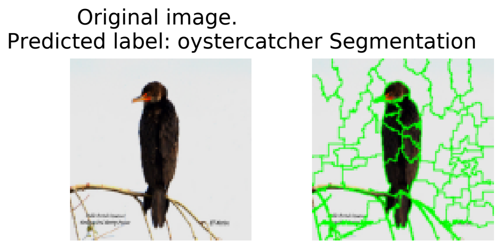

Default LIME
Manual color
Mean of the pixels
Mean of the superpixel's neighbors
Median among the mean of the superpixels
Median of the border's pixels
Median among the mean of the border's superpixels
Median among the mean of the superpixel's neighbors

(b) Replacement images

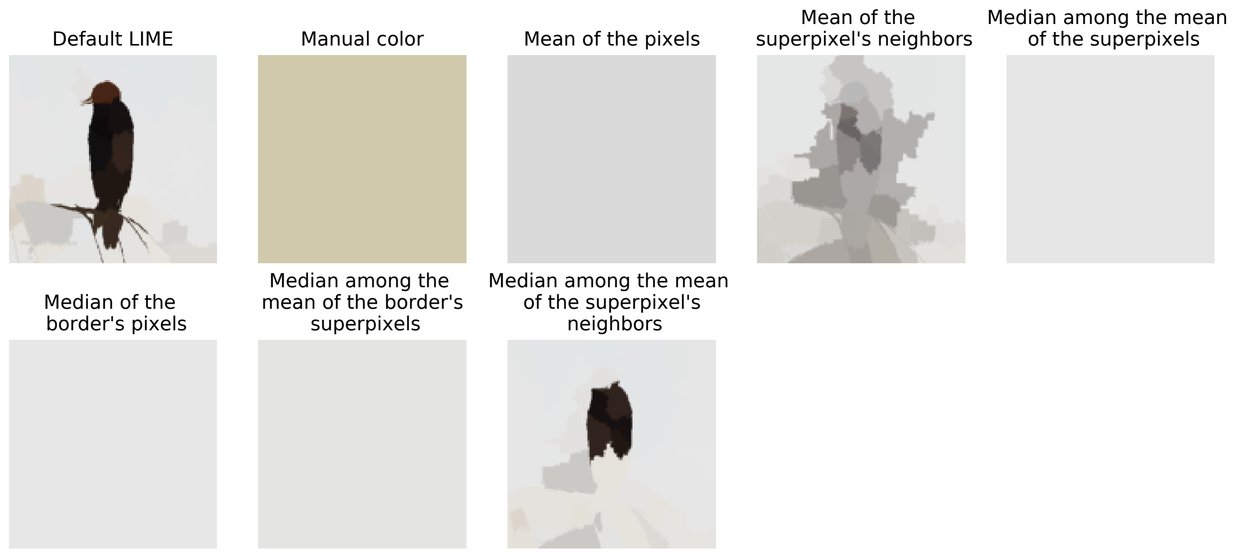

Default LIME
Manual color
Mean of the pixels
Mean of the superpixel's neighbors
Median among the mean of the superpixels
Median of the border's pixels
Median among the mean of the border's superpixels
Median among the mean of the superpixel's neighbors
LIME-G

(c) Explanations

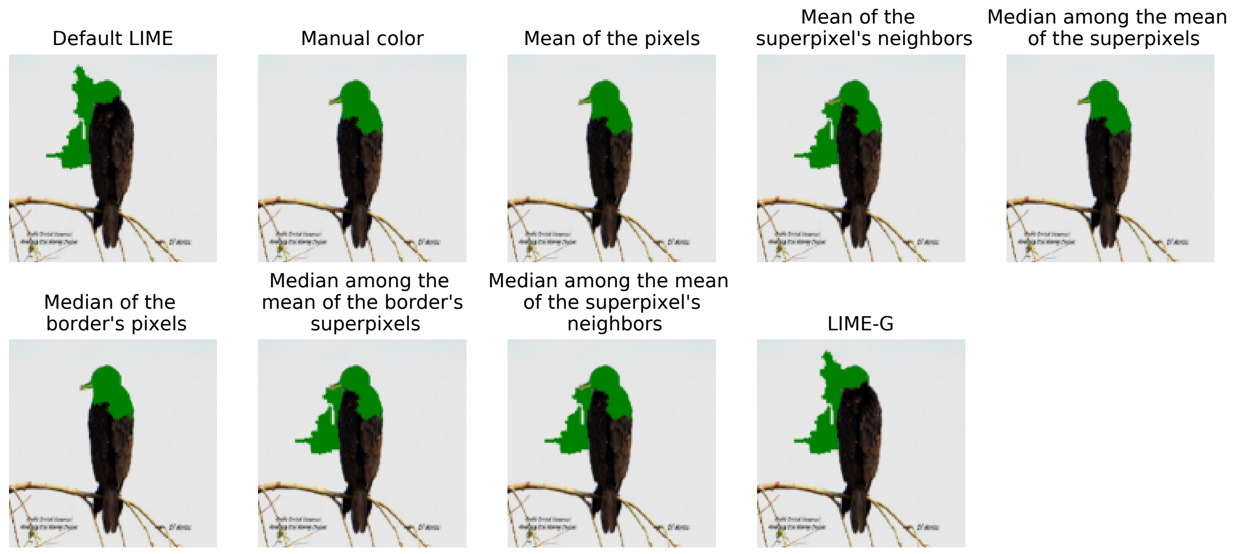
Figure 23: Comparison of LIME using different perturbed samples generation methods of an image where default LIME fails. Original image, superpixel segmentation, and replacement image and explanations (top 3 superpixels) for each perturbed samples generation method.



(a) Original image and segmentation

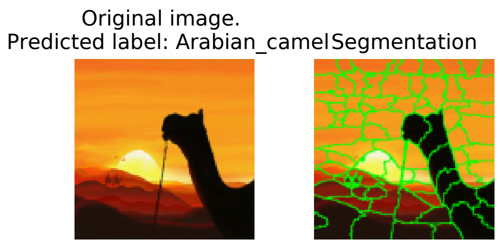

Default LIME
Manual color
Mean of the pixels
Mean of the superpixel's neighbors
Median among the mean of the superpixels
Median of the border's pixels
Median among the mean of the border's superpixels
Median among the mean of the superpixel's neighbors

(b) Replacement images

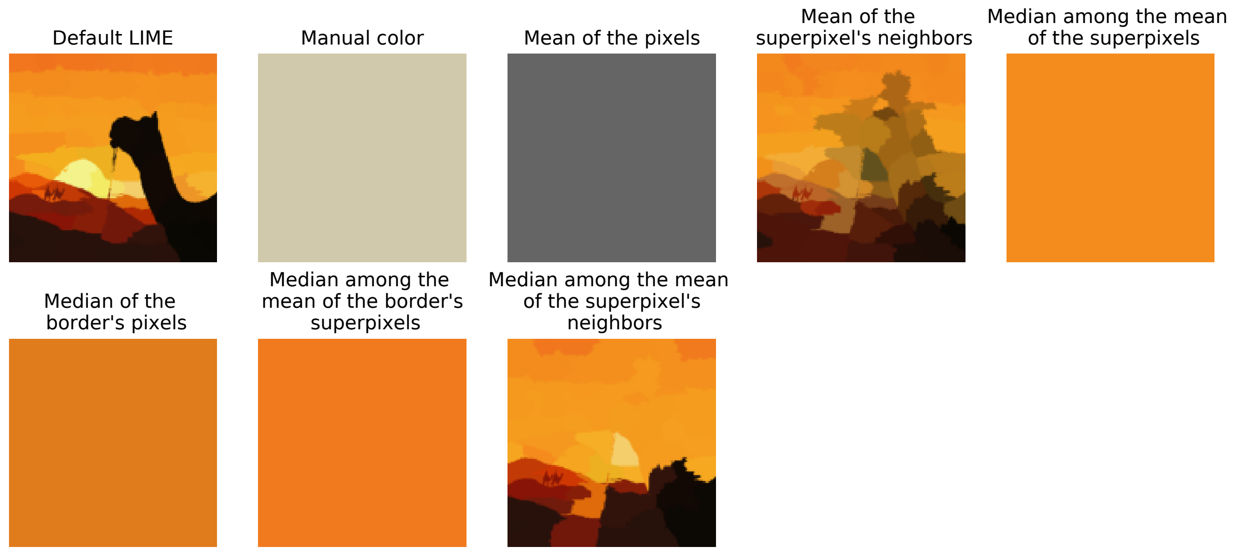

Default LIME
Manual color
Mean of the pixels
Mean of the superpixel's neighbors
Median among the mean of the superpixels
Median of the border's pixels
Median among the mean of the border's superpixels
Median among the mean of the superpixel's neighbors
LIME-G

(c) Explanations

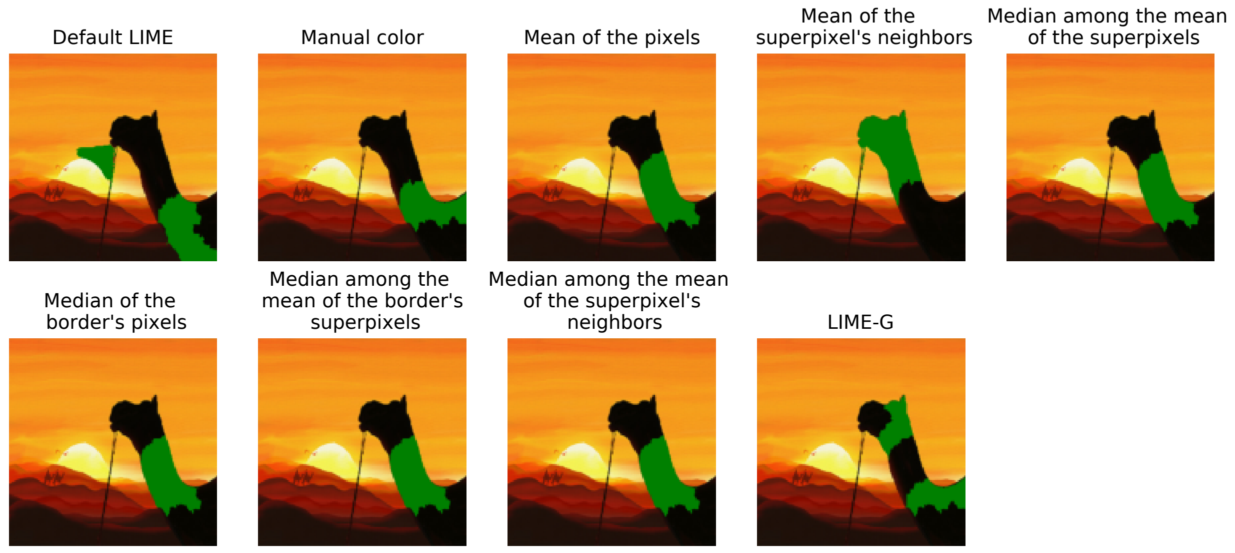
Figure 24: Comparison of LIME using different perturbed samples generation methods of an image where default LIME fails. Original image, superpixel segmentation, and replacement image and explanations (top 3 superpixels) for each perturbed samples generation method.



(a) Original image and segmentation

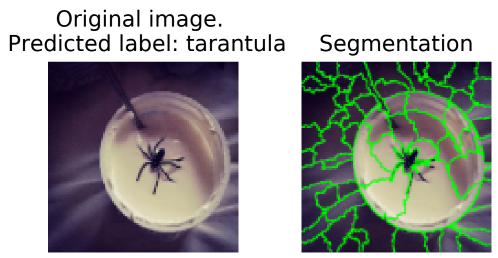

Default LIME
Manual color
Mean of the pixels
Mean of the superpixel's neighbors
Median among the mean of the superpixels
Median of the border's pixels
Median among the mean of the border's superpixels
Median among the mean of the superpixel's neighbors

(b) Replacement images

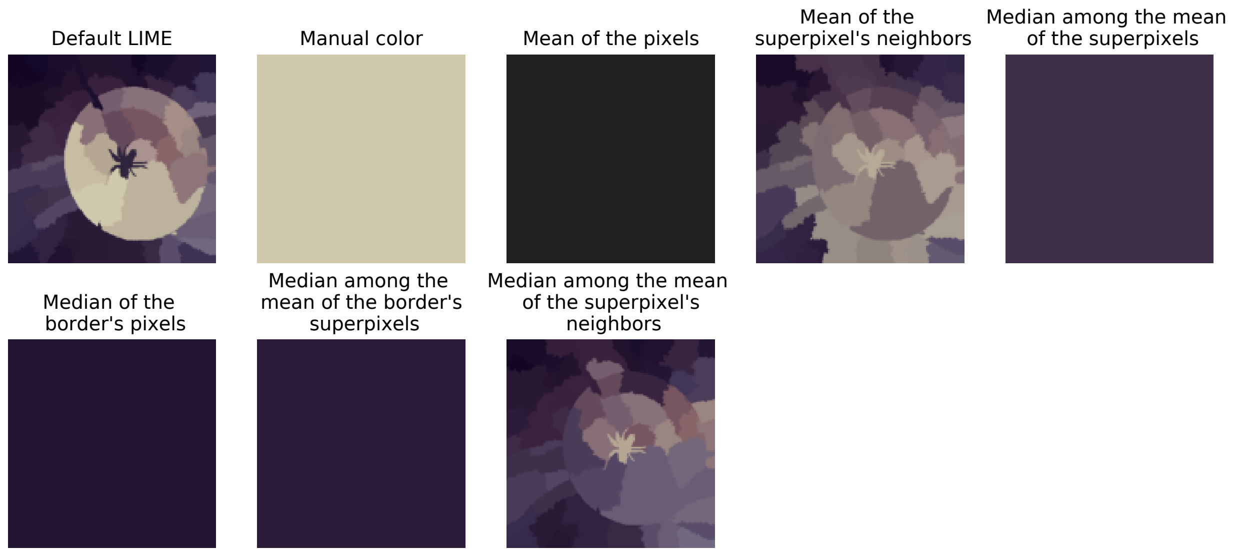

Default LIME
Manual color
Mean of the pixels
Mean of the superpixel's neighbors
Median among the mean of the superpixels
Median of the border's pixels
Median among the mean of the border's superpixels
Median among the mean of the superpixel's neighbors
LIME-G

(c) Explanations

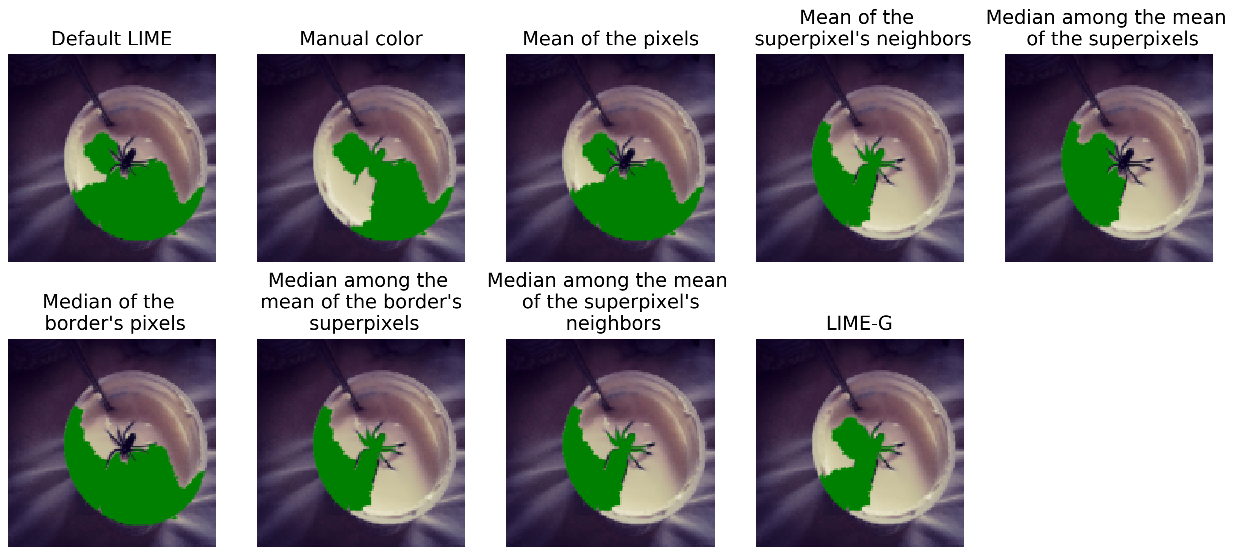
Figure 25: Comparison of LIME using different perturbed samples generation methods of an image where default LIME fails. Original image, superpixel segmentation, and replacement image and explanations (top 3 superpixels) for each perturbed samples generation method.



(a) Original image and segmentation

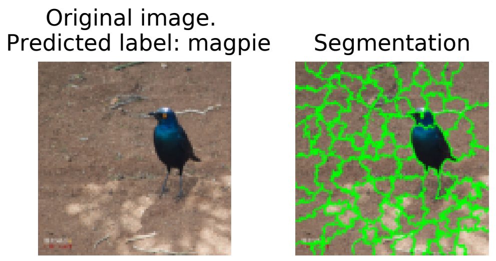

Default LIME
Manual color
Mean of the pixels
Mean of the superpixel's neighbors
Median among the mean of the superpixels
Median of the border's pixels
Median among the mean of the border's superpixels
Median among the mean of the superpixel's neighbors

(b) Replacement images

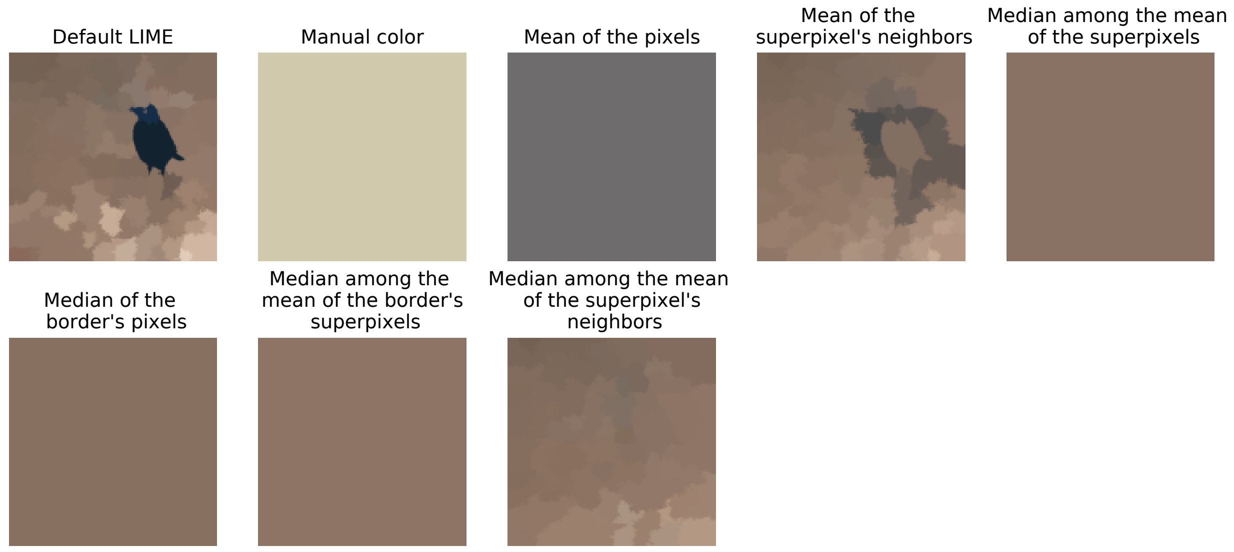

Default LIME
Manual color
Mean of the pixels
Mean of the superpixel's neighbors
Median among the mean of the superpixels
Median of the border's pixels
Median among the mean of the border's superpixels
Median among the mean of the superpixel's neighbors
LIME-G

(c) Explanations

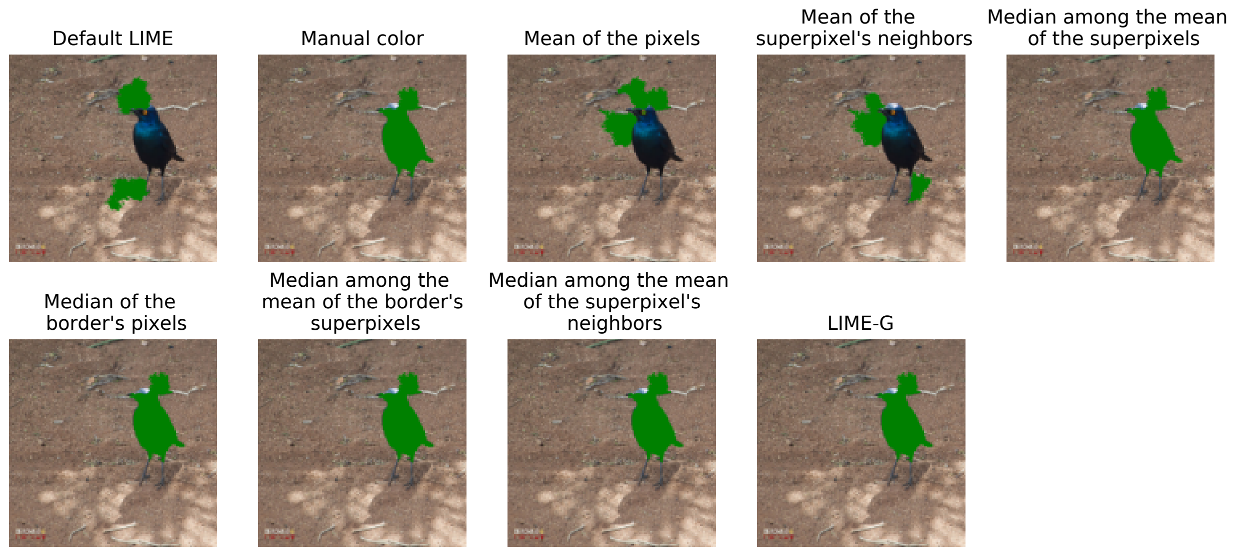
Figure 26: Comparison of LIME using different perturbed samples generation methods of an image where default LIME fails. Original image, superpixel segmentation, and replacement image and explanations (top 3 superpixels) for each perturbed samples generation method.



(a) Original image and segmentation

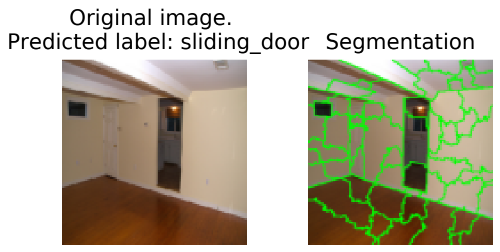

Default LIME
Manual color
Mean of the pixels
Mean of the superpixel's neighbors
Median among the mean of the superpixels
Median of the border's pixels
Median among the mean of the border's superpixels
Median among the mean of the superpixel's neighbors

(b) Replacement images

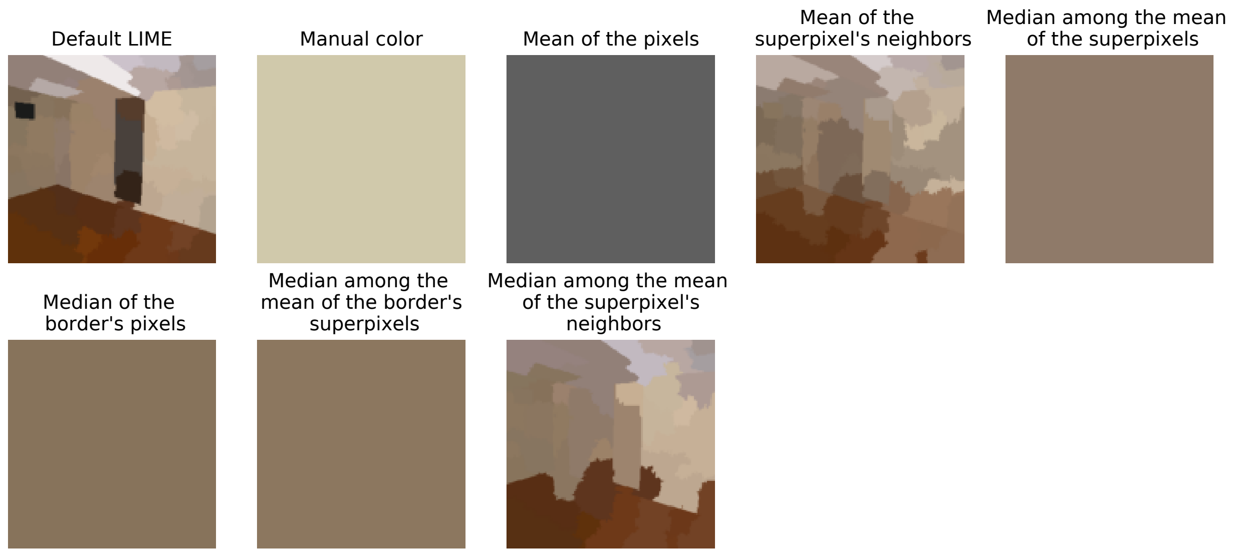

Default LIME
Manual color
Mean of the pixels
Mean of the superpixel's neighbors
Median among the mean of the superpixels
Median of the border's pixels
Median among the mean of the border's superpixels
Median among the mean of the superpixel's neighbors
LIME-G

(c) Explanations

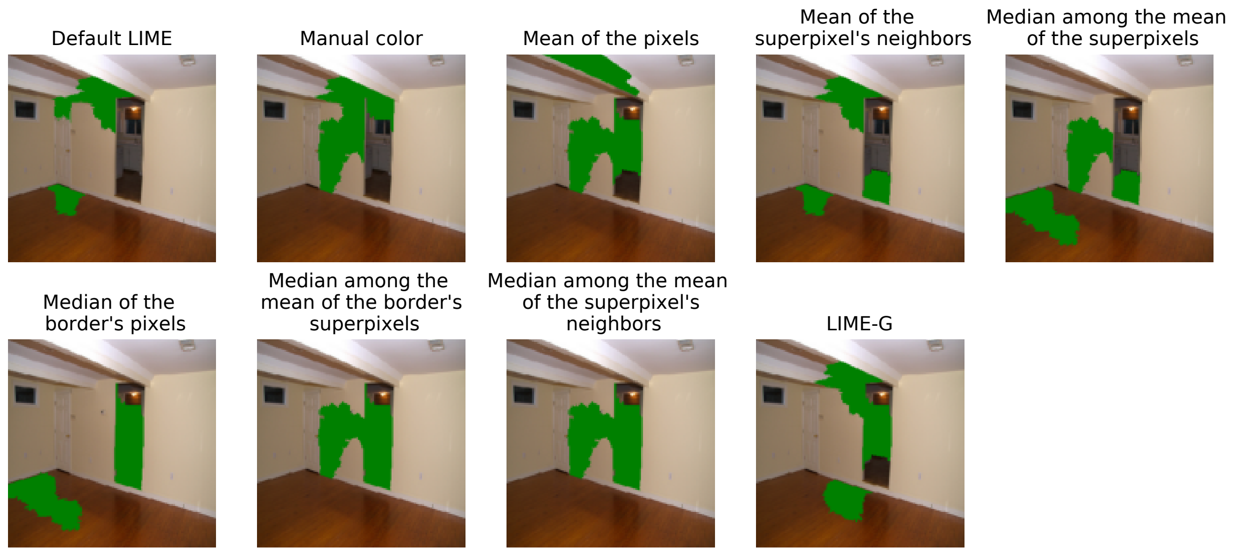
Figure 27: Comparison of LIME using different perturbed samples generation methods of an image where default LIME fails. Original image, superpixel segmentation, and replacement image and explanations (top 3 superpixels) for each perturbed samples generation method.



(a) Original image and segmentation

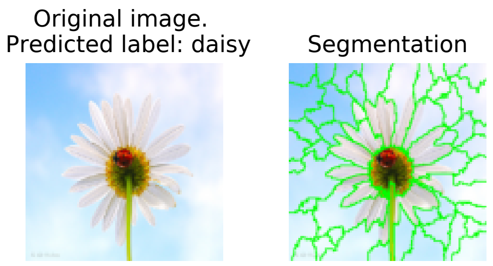

Default LIME
Manual color
Mean of the pixels
Mean of the superpixel's neighbors
Median among the mean of the superpixels
Median of the border's pixels
Median among the mean of the border's superpixels
Median among the mean of the superpixel's neighbors

(b) Replacement images

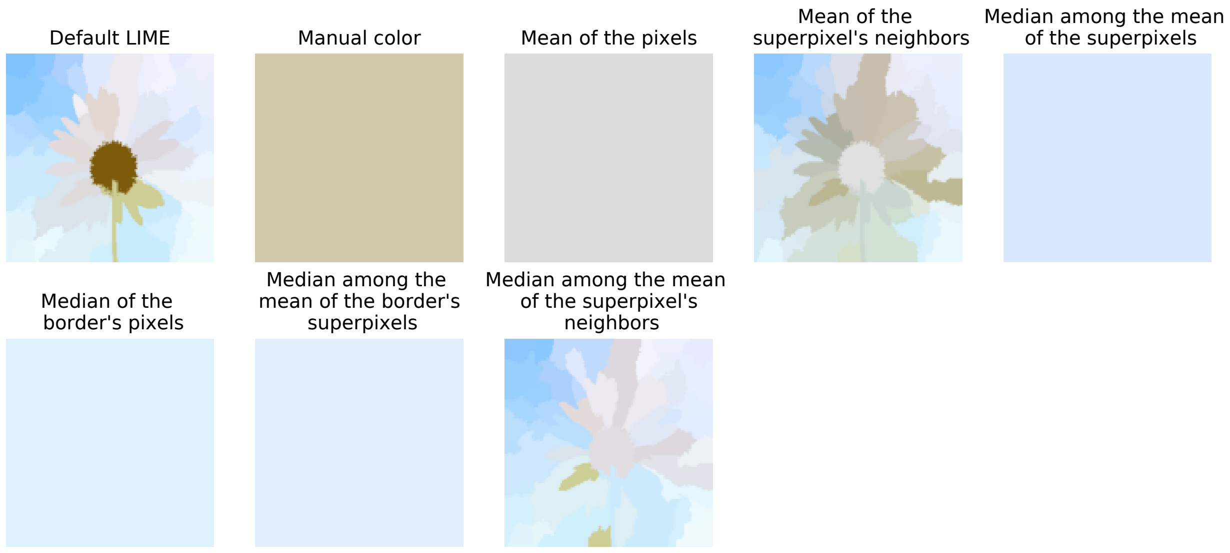

Default LIME
Manual color
Mean of the pixels
Mean of the superpixel's neighbors
Median among the mean of the superpixels
Median of the border's pixels
Median among the mean of the border's superpixels
Median among the mean of the superpixel's neighbors
LIME-G

(c) Explanations

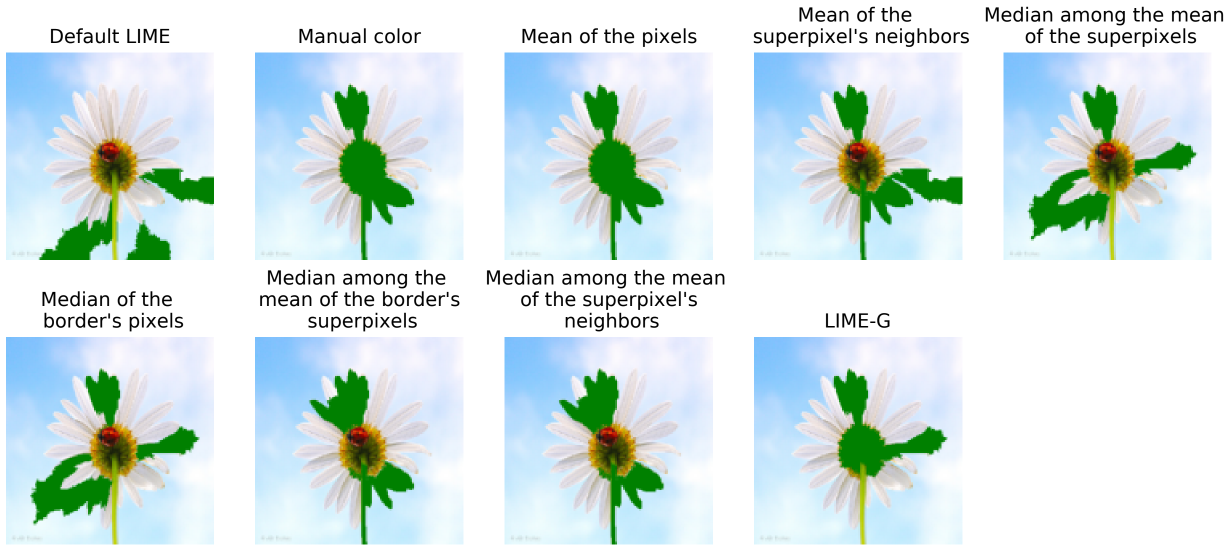
Figure 28: Comparison of LIME using different perturbed samples generation methods of an image where default LIME fails. Original image, superpixel segmentation, and replacement image and explanations (top 3 superpixels) for each perturbed samples generation method.



(a) Original image and segmentation

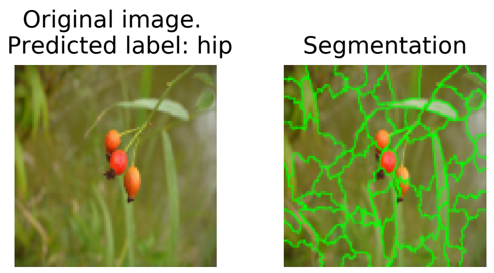

Default LIME
Manual color
Mean of the pixels
Mean of the superpixel's neighbors
Median among the mean of the superpixels
Median of the border's pixels
Median among the mean of the border's superpixels
Median among the mean of the superpixel's neighbors

(b) Replacement images

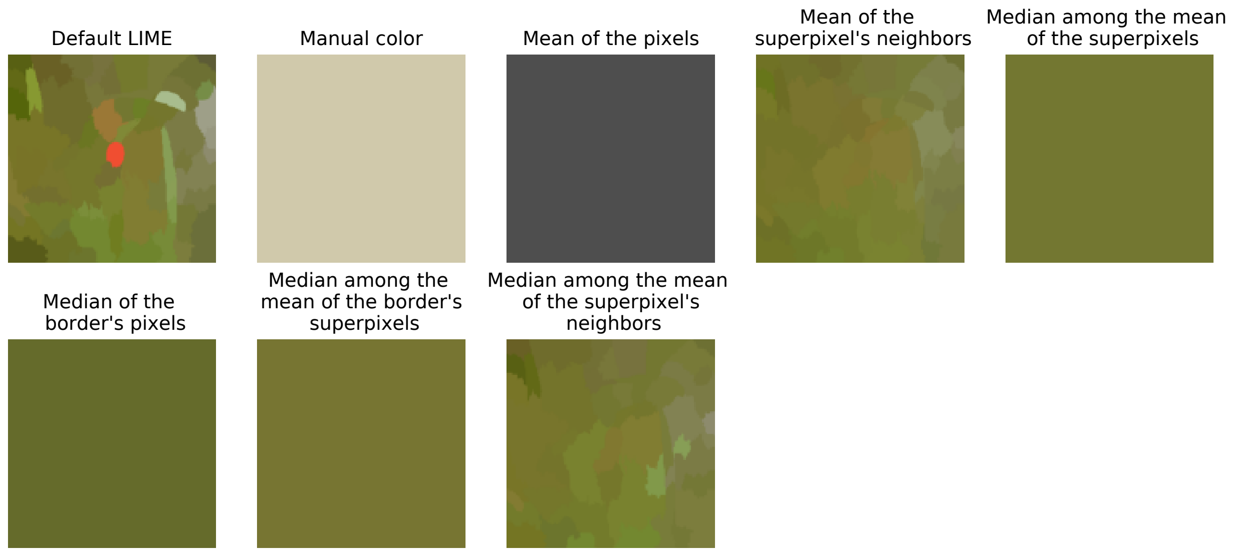

Default LIME
Manual color
Mean of the pixels
Mean of the superpixel's neighbors
Median among the mean of the superpixels
Median of the border's pixels
Median among the mean of the border's superpixels
Median among the mean of the superpixel's neighbors
LIME-G

(c) Explanations

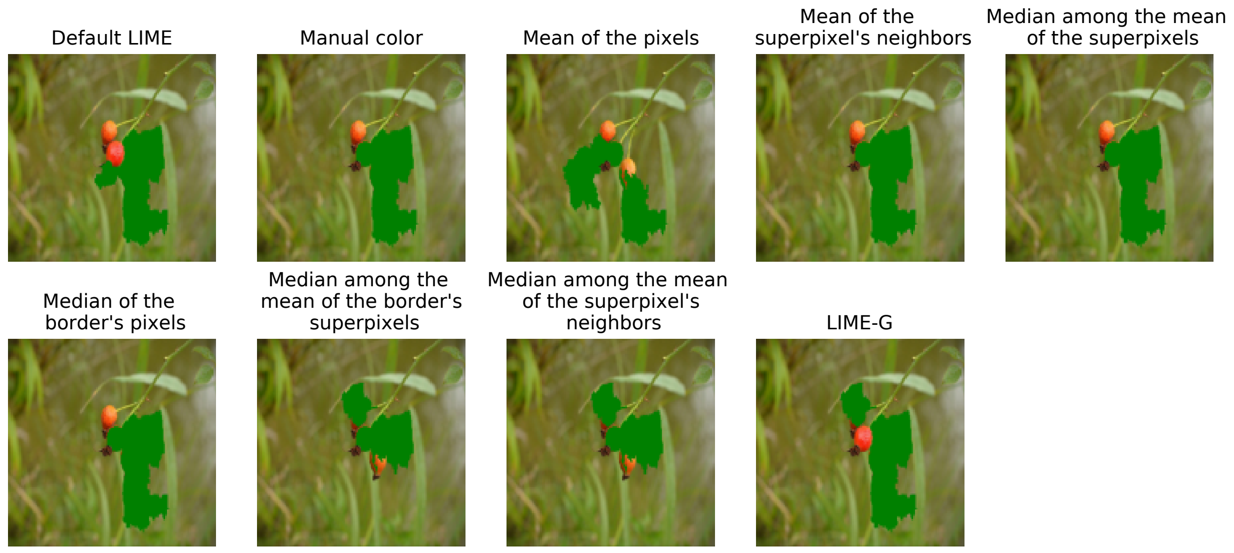
Figure 29: Comparison of LIME using different perturbed samples generation methods of an image where default LIME fails. Original image, superpixel segmentation, and replacement image and explanations (top 3 superpixels) for each perturbed samples generation method.



(a) Original image and segmentation

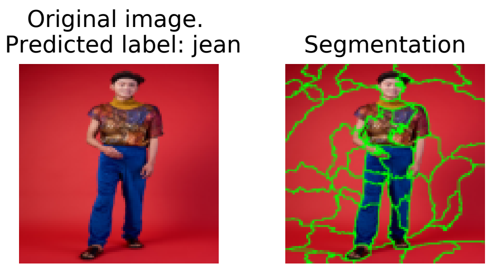

Default LIME
Manual color
Mean of the pixels
Mean of the superpixel's neighbors
Median among the mean of the superpixels
Median of the border's pixels
Median among the mean of the border's superpixels
Median among the mean of the superpixel's neighbors

(b) Replacement images

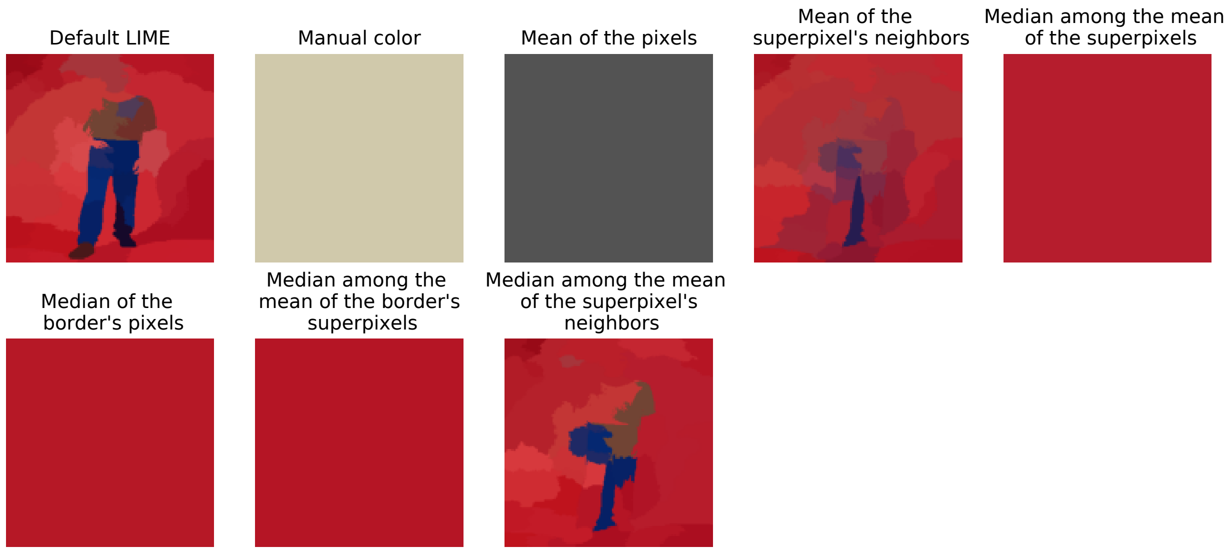

Default LIME
Manual color
Mean of the pixels
Mean of the superpixel's neighbors
Median among the mean of the superpixels
Median of the border's pixels
Median among the mean of the border's superpixels
Median among the mean of the superpixel's neighbors
LIME-G

(c) Explanations

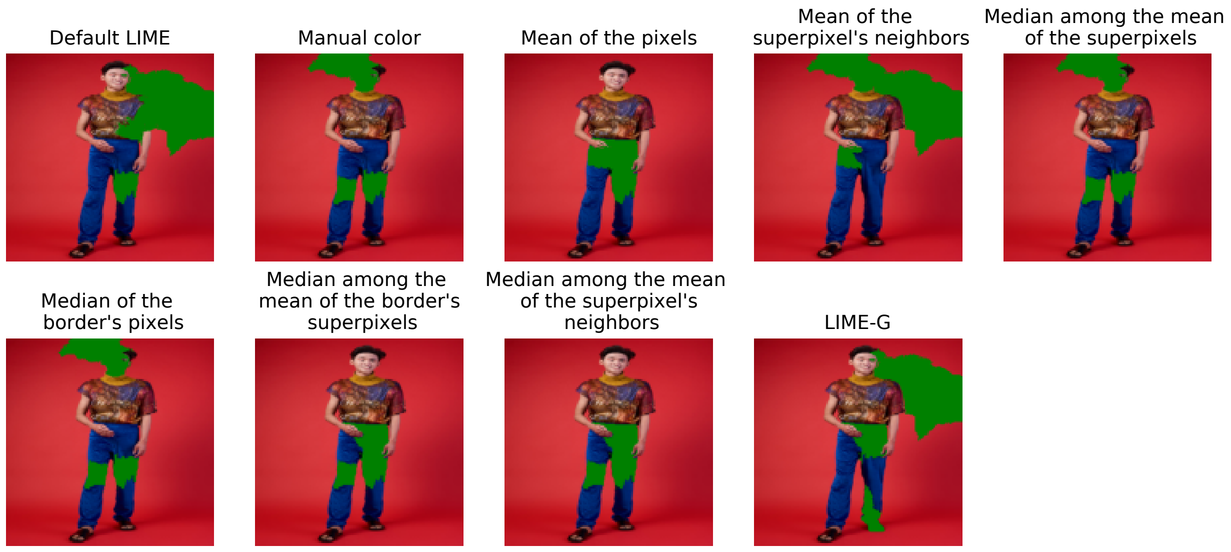
Figure 30: Comparison of LIME using different perturbed samples generation methods of an image where default LIME fails. Original image, superpixel segmentation, and replacement image and explanations (top 3 superpixels) for each perturbed samples generation method.



(a) Original image and segmentation


Default LIME
Manual color
Mean of the pixels
Mean of the superpixel's neighbors
Median among the mean of the superpixels
Median of the border's pixels
Median among the mean of the border's superpixels
Median among the mean of the superpixel's neighbors

(b) Replacement images


Default LIME
Manual color
Mean of the pixels
Mean of the superpixel's neighbors
Median among the mean of the superpixels
Median of the border's pixels
Median among the mean of the border's superpixels
Median among the mean of the superpixel's neighbors
LIME-G

(c) Explanations

Figure 31: Comparison of LIME using different perturbed samples generation methods of an image where default LIME fails. Original image, superpixel segmentation, and replacement image and explanations (top 3 superpixels) for each perturbed samples generation method.

9.2 Variation in the number of superpixels

We observed that the band aid example was challenging for all the perturbed sample generation methods that require the creation of a replacement image. The whole band aid area is enclosed by a superpixel. This could cause that, regardless of the replacement color used to *turn off* the superpixel, it would still look like a band aid for the classifier. For instance, even if we use a skin-like color to *turn off* the superpixel, it makes it resemble a skin-tone band-aid. Therefore, one possible solution could be to force the band-aid area to be fragmented among several superpixels. To achieve this, we modified the parameters of the *quickshift* algorithm. More specifically, we change the size of the kernel. The smaller the size, the greater the number of superpixels.

If we reduce the number of superpixels (see Figure 32 for kernel size 5 and 4.5), the band aid superpixel is selected among the top 10. Nevertheless, the heatmap indicates that the most important region is the one around the band-aid. Hence, given that there are fewer superpixels, it is easier for the superpixel containing the band aid to be in the top 10. Increasing the number of superpixels (kernel size 1.9 and 2.3), and therefore fragmenting the band-aid among more than one superpixel, does not help in recognizing the most important area neither.

We decided to vary the number of superpixels in an image that is not challenging for LIME (Figure 33). Increasing the number of superpixels does not help either is this example. We can suspect that too many superpixels disables LIME from working properly because the surrogate model that LIME uses is ridge regression, which, as other regression models, needs $n \gg d$ to build a satisfying model.

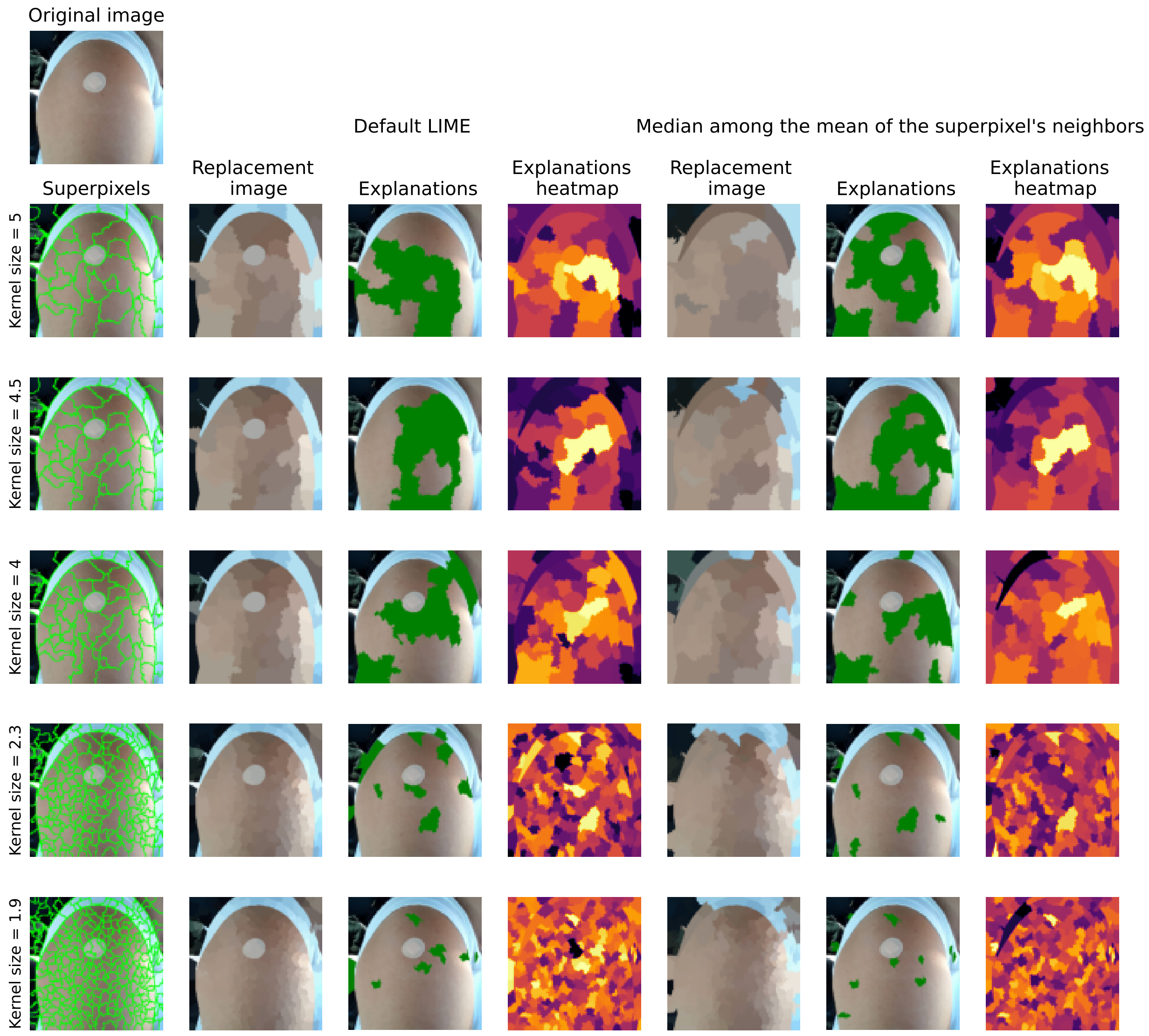


Figure 32: Comparison of LIME results varying the number of superpixels for the band-aid example. Each row presents the results for a different kernel on the *quickshift* algorithm, which results in an increase (or decrease) in the number of superpixels. We display the replacement image, explanations (top 10 superpixels highlighted in green), and the heatmap (superpixels colored according their coefficients) for default LIME and LIME using *median among the mean of the superpixel's neighbors*.

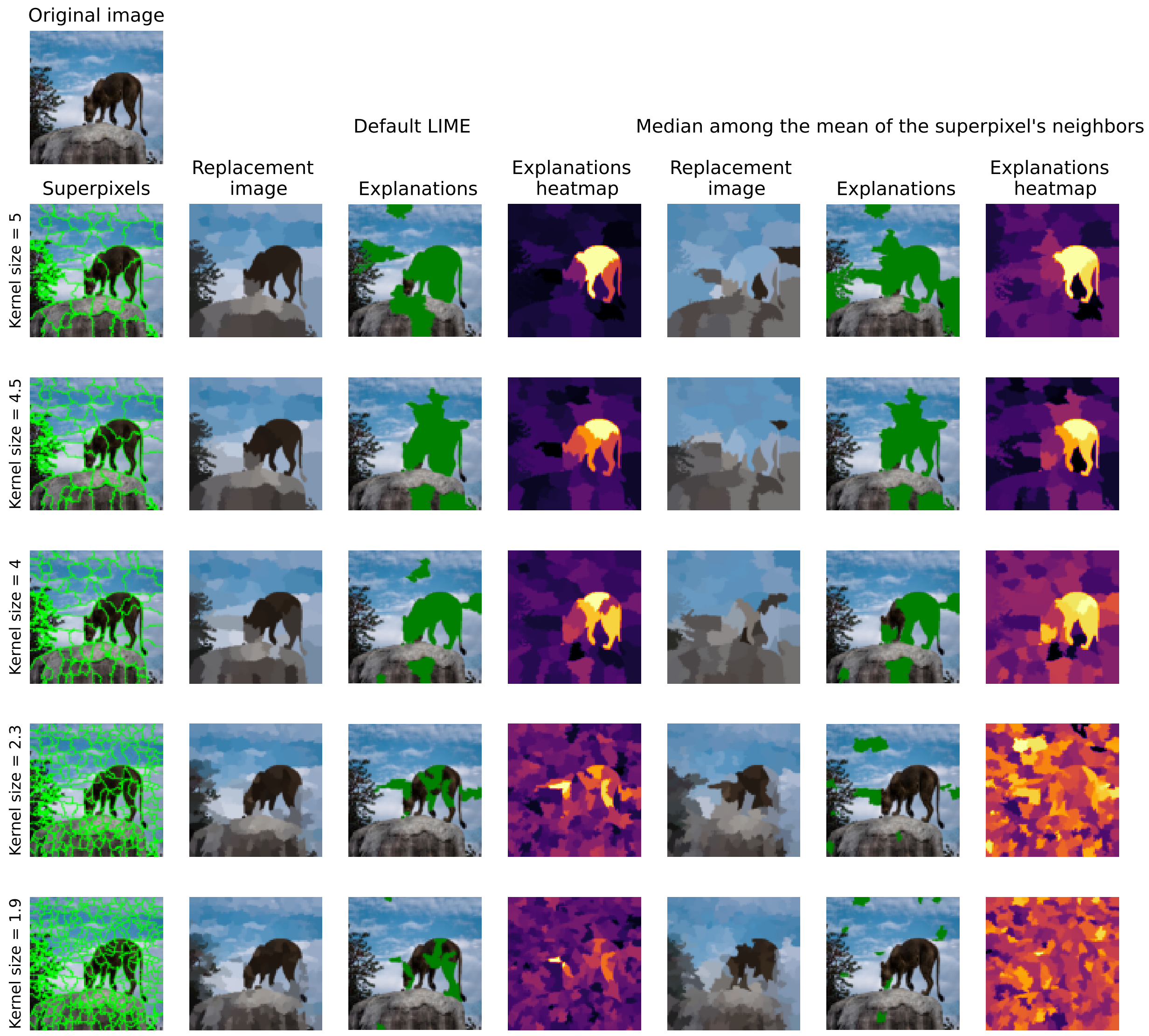


Figure 33: Comparison of LIME results varying the number of superpixels for the lion example. Each row presents the results for a different kernel on the *quickshift* algorithm, which results in an increase (or decrease) in the number of superpixels. We display the replacement image, explanations (top 10 superpixels highlighted in green), and the heatmap (superpixels colored according their coefficients) for default LIME and LIME using *median among the mean of the superpixel's neighbors*.

References

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 1135–1144, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939778. event-place: San Francisco, California, USA.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-Precision Model-Agnostic Explanations. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), April 2018.

Scott M. Lundberg and Su-In Lee. A Unified Approach to Interpreting Model Predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 4768–4777, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 978-1-5108-6096-4. event-place: Long Beach, California, USA.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. In *Workshop at International Conference on Learning Representations*, 2014.

Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning Important Features Through Propagating Activation Differences. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3145–3153. PMLR, August 2017.

Pantelis Linardatos, Vasilis Papastefanopoulos, and Sotiris Kotsiantis. Explainable AI: A Review of Machine Learning Interpretability Methods. *Entropy*, 23(1), 2021. ISSN 1099-4300. doi: 10.3390/e23010018.

Chirag Agarwal and Anh Nguyen. Explaining image classifiers by removing input features using generative models. In *Proceedings of the Asian Conference on Computer Vision (ACCV)*, November 2020.

Christoph Molnar. *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable.* 2019.

Damien Garreau and Dina Mardaoui. What does LIME really see in images? In *ICML 2021 - 38th International Conference on Machine Learning*, virtual, United States, July 2021.

Damien Garreau and Ulrike von Luxburg. Looking Deeper into Tabular LIME, 2020.

Dina Mardaoui and Damien Garreau. An Analysis of LIME for Text Data. In *AISTATS 2021 - 24th International Conference on Artificial Intelligence and Statistics*, Vienne, Austria, April 2021.

Andrea Vedaldi and Stefano Soatto. Quick Shift and Kernel Methods for Mode Seeking. In David Forsyth, Philip Torr, and Andrew Zisserman, editors, *Computer Vision – ECCV 2008*, pages 705–718, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-88693-8.

Arthur E. Hoerl and Robert W. Kennard. Ridge Regression: Biased Estimation for Nonorthogonal Problems. *Technometrics*, 12(1):55–67, 1970.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the Inception Architecture for Computer Vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, Los Alamitos, CA, USA, June 2016. IEEE Computer Society. ISSN: 1063-6919.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic Attribution for Deep Networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 3319–3328. JMLR.org, 2017. event-place: Sydney, NSW, Australia.

J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. S. Huang. Generative Image Inpainting with Contextual Attention. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5505–5514, Los Alamitos, CA, USA, June 2018. IEEE Computer Society. doi: 10.1109/CVPR.2018.00577.

Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. SLIC Superpixels Compared to State-of-the-Art Superpixel Methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(11):2274–2282, 2012. doi: 10.1109/TPAMI.2012.120.

Shengyu Zhao, Jonathan Cui, Yilun Sheng, Yue Dong, Xiao Liang, Eric I.-Chao Chang, and Yan Xu. Large Scale Image Completion via Co-Modulated Generative Adversarial Networks. *CoRR*, abs/2103.10428, 2021. _eprint: 2103.10428.

Jingyuan Li, Ning Wang, Lefei Zhang, Bo Du, and Dacheng Tao. Recurrent Feature Reasoning for Image Inpainting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020a.

Hongyu Liu, Bin Jiang, Yibing Song, Wei Huang, and Chao Yang. Rethinking Image Inpainting via a Mutual Encoder-Decoder with Feature Equalizations. *CoRR*, abs/2007.06929, 2020. _eprint: 2007.06929.

Gourav Wadhwa, Abhinav Dhall, Subrahmanyam Murala, and Usman Tariq. Hyperrealistic Image Inpainting With Hypergraphs. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 3912–3921, January 2021.

Chu-Tak Li, Wan-Chi Siu, Zhi-Song Liu, Li-Wen Wang, and Daniel Pak-Kong Lun. DeepGIN: Deep Generative Inpainting Network for Extreme Image Inpainting. *CoRR*, abs/2008.07173, 2020b. _eprint: 2008.07173.

Alexandru Telea. An Image Inpainting Technique Based on the Fast Marching Method. *Journal of Graphics Tools*, 9(1):23–34, 2004. doi: 10.1080/10867651.2004.10487596. Publisher: Taylor & Francis.

David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert Müller. How to Explain Individual Classification Decisions. *Journal of Machine Learning Research*, 11 (61):1803–1831, 2010. URL `http://jmlr.org/papers/v11/baehrens10a.html`.

Ruth Fong and Andrea Vedaldi. Interpretable Explanations of Black Boxes by Meaningful Perturbation. *CoRR*, abs/1704.03296, 2017. _eprint: 1704.03296.