Ex11.InterpretableML.with.LIME

January 16, 2024

1 11th exercise: Interpretable Machine Learning with LIME for Image Classification

- Course: AML
- Lecturer: Gernot Heisenberg
- Author of notebook: Finn Heydemann
- Date: 18.12.2023

GENERAL NOTE 1: Please make sure you are reading the entire notebook, since it contains a lot of information on your tasks (e.g. regarding the set of certain paramaters or a specific computational trick), and the written mark downs as well as comments contain a lot of information on how things work together as a whole.
GENERAL NOTE 2: * Please, when commenting source code, just use English language only. * When describing an observation please use English language, too. * This applies to all exercises throughout this course.

1.0.1 DESCRIPTION:

LIME stands for Local Interpretable Model-agnostic Explanations and was developed by Ribeiro et.al. in 2016 (https://arxiv.org/abs/1602.04938).

In this notebook LIME is going to be used to generate explanations for an image classification task. The basic idea applying LIME is to understand why a deep neural network predicts that an instance (image) belongs to a certain class (labrador in this case). This notebook is based on previous work by Cristian Arteaga, arteagac.github.io

---

1.0.2 TASKS:

The tasks that you need to work on within this notebook are always indicated below as bullet points. If a task is more challenging and consists of several steps, this is indicated as well. Make

sure you have worked down the task list and commented your doings. This should be done by using markdown. Make sure you don't forget to specify your name and your matriculation number in the notebook.

YOUR TASKS in this exercise are as follows: 1. import the notebook to Google Colab or use your local machine. 2. make sure you specified you name and your matriculation number in the header below my name and date. * set the date too and remove mine. 3. read the entire notebook carefully * add comments whereever you feel it necessary for better understanding * run the notebook for the first time. 4. test your own images to obtain explanations for your classification tasks. 5. change some hyperparameters 6. describe your observations and the LIME performance ———————————

1.0.3 Imports

Import all necessary python utilities for manipulation of images, plotting and numerical analysis.

[1]:
```
import numpy as np
import tensorflow as tf
import tensorflow.keras
from   tensorflow.keras.applications.imagenet_utils import decode_predictions
import skimage.io
import skimage.segmentation
import copy
import sklearn
import sklearn.metrics
from   sklearn.linear_model import LinearRegression
import warnings

np.random.seed(666)
```

```
2023-12-18 16:06:22.705301: I tensorflow/core/platform/cpu_feature_guard.cc:182]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: SSE4.1 SSE4.2 AVX AVX2 AVX_VNNI FMA, in
other operations, rebuild TensorFlow with the appropriate compiler flags.
```

1.0.4 InceptionV3 initialization

We are going to use the pre-trained InceptionV3 model available in Keras.

[2]:
```
warnings.filterwarnings('ignore')
inceptionV3_model = tensorflow.keras.applications.inception_v3.InceptionV3()␣
 ↪#Load pretrained model
```

```
2023-12-18 16:06:23.952174: I
tensorflow/core/common_runtime/process_util.cc:146] Creating new thread pool
with default inter op setting: 2. Tune using inter_op_parallelism_threads for
best performance.
```

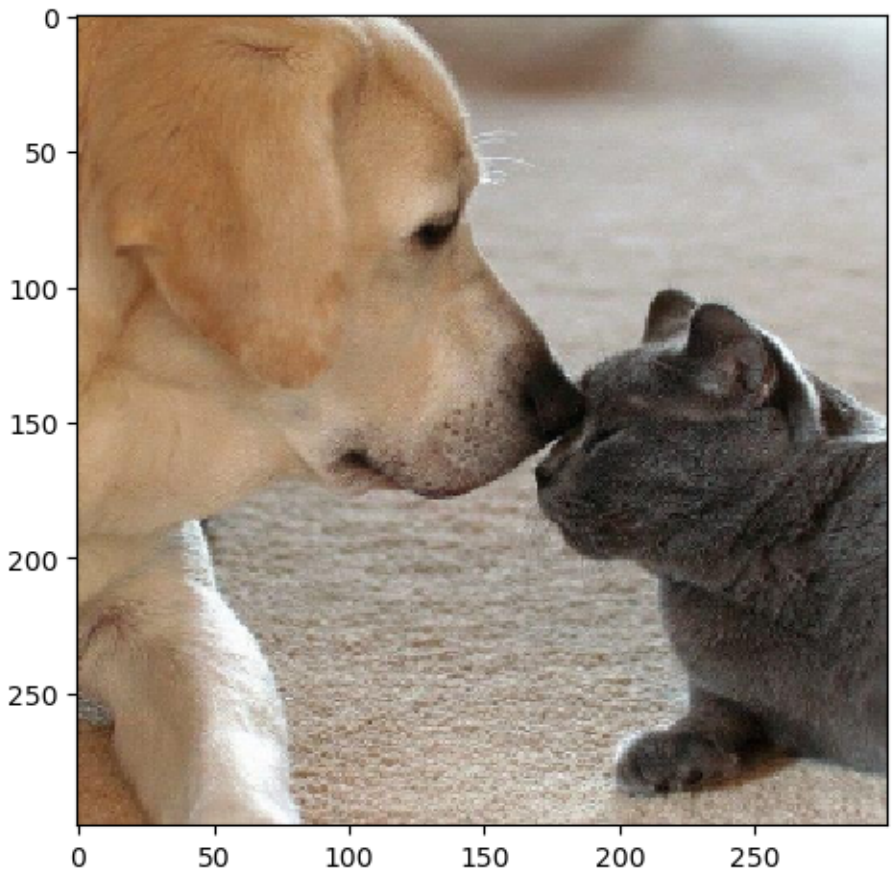
```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels.h5
96112376/96112376 [==============================] - 42s 0us/step
```

1.0.5 Read and pre-process image

The instance to be explained (image) is resized and pre-processed to be suitable for Inception V3. This image is saved in the variable `Xi`.

[3]:
```
Xi = skimage.io.imread("./data/cat-and-dog.jpg")
Xi = skimage.transform.resize(Xi, (299,299))
Xi = (Xi - 0.5)*2 #Inception pre-processing
skimage.io.imshow(Xi/2+0.5) # Show image before inception preprocessing
```

[3]:
```
<matplotlib.image.AxesImage at 0x7f91f879d910>
```



1.0.6 Predict the class of the input image

The Inception V3 model is used to predict the class of the image. The output of the classification is a vector of 1000 proabilities of beloging to each class available in Inception V3. The description

of these classes is shown and it can be seen that the "Labrador Retriever" is the top class for the given image.

[4]:
```
np.random.seed(666)
preds = inceptionV3_model.predict(Xi[np.newaxis,:,:,:])
decode_predictions(preds)[0] #Top 5 classes
```

```
1/1 [==============================] - 1s 711ms/step
Downloading data from https://storage.googleapis.com/download.tensorflow.org/dat
a/imagenet_class_index.json
35363/35363 [==============================] - 0s 6us/step
```

[4]:
```
[('n02099712', 'Labrador_retriever', 0.8221335),
 ('n02099601', 'golden_retriever', 0.015566387),
 ('n02093428', 'American_Staffordshire_terrier', 0.009475612),
 ('n02108422', 'bull_mastiff', 0.00831723),
 ('n02109047', 'Great_Dane', 0.00783028)]
```

The indexes (positions) of the top 5 classes are saved in the variable `top_pred_classes`

[5]:
```
top_pred_classes = preds[0].argsort()[-5:][::-1]
top_pred_classes                #Index of top 5 classes
```

[5]:
```
array([208, 207, 180, 243, 246])
```

1.1 LIME explanations

The following figure illustrates the basic idea behind LIME. The figure shows light and dark gray areas which are the decision boundaries for the classes for each (x1,x2) pairs in the dataset. LIME is able to provide explanations for the predictions of an individual record (blue dot). The explanations are created by generating a new dataset of perturbations around the instance to be explained (colored markers around the blue dot). The output or class of each generated perturbation is predicted with the machine-learning model (colored markers inside and outside the decision boundaries). The importance of each perturbation is determined by measuring its distance from the original instance to be explained. These distances are converted to weights by mapping the distances to a zero-one scale using a kernel function (see color scale for the weights). All this information: the new generated dataset, its class predictions and its weights are used to fit a simpler model, such as a linear model (blue line), that can be interpreted. The attributes of the simpler model, coefficients for the case of a linear model, are then used to generate explanations.

A detailed explanation of each step is shown below.

1.1.1 Step 1: Create perturbations of image

For the case of image explanations, perturbations will be generated by turning on and off some of the superpixels in the image.

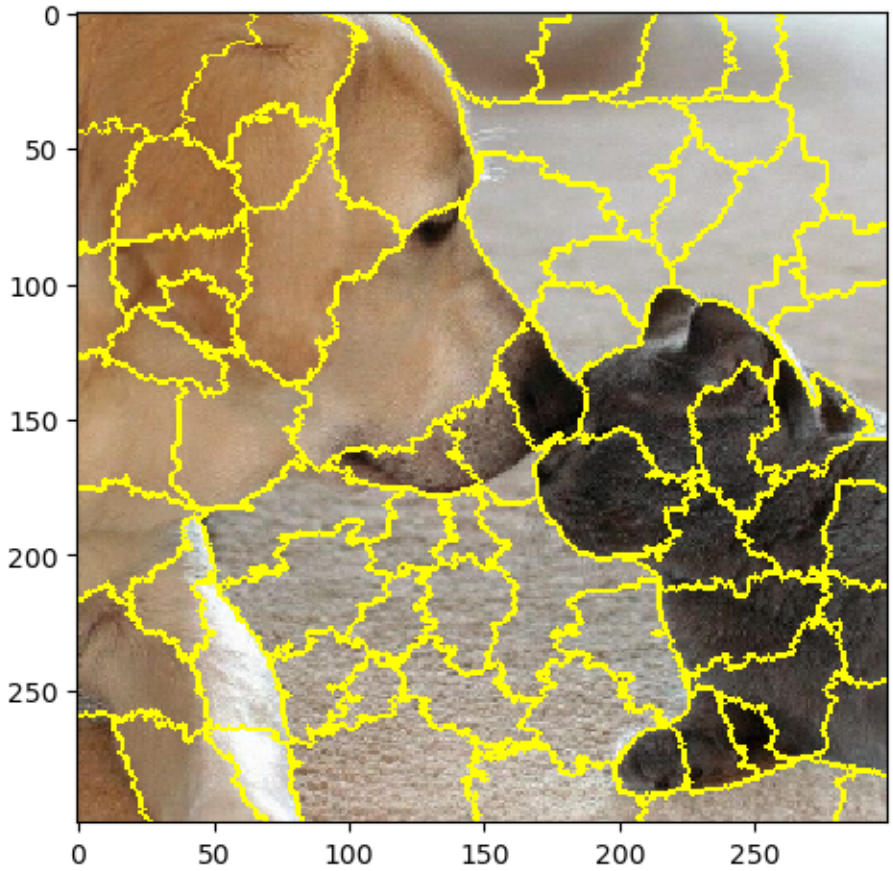
Extract super-pixels from image Superpixels are generated using the quickshift segmentation algorithm. It can be noted that for the given image, 69 superpixels were generated. The generated superpixels are shown in the image below.

[6]:
```
superpixels = skimage.segmentation.quickshift(Xi, kernel_size=4,max_dist=200,␣
↪ratio=0.2)
num_superpixels = np.unique(superpixels).shape[0]
num_superpixels
```

[6]: 69

[7]:
```
skimage.io.imshow(skimage.segmentation.mark_boundaries(Xi/2+0.5, superpixels))
```

[7]: <matplotlib.image.AxesImage at 0x7f91f8525e10>



Create random perturbations In this example, 150 perturbations were used. However, for real life applications, a larger number of perturbations will produce more reliable explanations. Random zeros and ones are generated and shaped as a matrix with perturbations as rows and superpixels as columns. An example of a perturbation (the first one) is show below. Here, `1` represents that a

superpixel is on and 0 represents it is off. Notice that the length of the shown vector corresponds to the number of superpixels in the image.

[8]:
```
num_perturb = 150
perturbations = np.random.binomial(1, 0.5, size=(num_perturb, num_superpixels))
perturbations[0] #Show example of perturbation
```

[8]:
```
array([1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1,
       0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0,
       0, 0, 1])
```

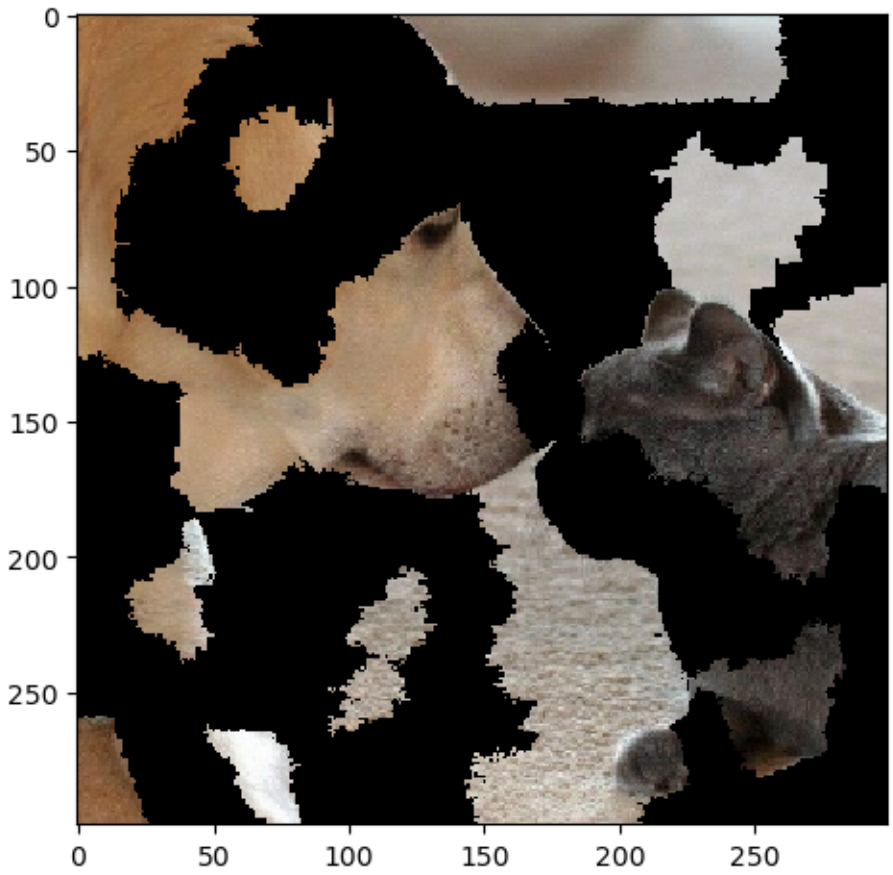
The following function `perturb_image` perturbs the given image (`img`) based on a perturbation vector (`perturbation`) and predefined superpixels (`segments`).

[9]:
```
def perturb_image(img,perturbation,segments):
    active_pixels = np.where(perturbation == 1)[0]
    mask = np.zeros(segments.shape)
    for active in active_pixels:
        mask[segments == active] = 1
    perturbed_image = copy.deepcopy(img)
    perturbed_image = perturbed_image*mask[:,:,np.newaxis]
    return perturbed_image
```

Let's use the previous function to see what a perturbed image would look like:

[10]:
```
skimage.io.imshow(perturb_image(Xi/2+0.5,perturbations[0],superpixels))
```

[10]:
```
<matplotlib.image.AxesImage at 0x7f91f0f63350>
```



1.1.2 Step 2: Use ML classifier to predict classes of new generated images

This is computationally the most expensive step in LIME because a prediction for each perturbed image is computed. From the shape of the predictions, we can see for each of the perturbations the output probability for each of the 1000 classes in Inception V3.

[11]:
```
predictions = []
for pert in perturbations:
    perturbed_img = perturb_image(Xi,pert,superpixels)
    pred = inceptionV3_model.predict(perturbed_img[np.newaxis,:,:,:])
    predictions.append(pred)

predictions = np.array(predictions)
predictions.shape
```

```
1/1 [==============================] - 0s 161ms/step
1/1 [==============================] - 0s 186ms/step
1/1 [==============================] - 0s 168ms/step
1/1 [==============================] - 0s 193ms/step
1/1 [==============================] - 0s 119ms/step
```

```
1/1 [==============================] - 0s 151ms/step
1/1 [==============================] - 0s 220ms/step
1/1 [==============================] - 0s 187ms/step
1/1 [==============================] - 0s 147ms/step
1/1 [==============================] - 0s 175ms/step
1/1 [==============================] - 0s 149ms/step
1/1 [==============================] - 0s 188ms/step
1/1 [==============================] - 0s 170ms/step
1/1 [==============================] - 0s 171ms/step
1/1 [==============================] - 0s 167ms/step
1/1 [==============================] - 0s 215ms/step
1/1 [==============================] - 0s 136ms/step
1/1 [==============================] - 0s 115ms/step
1/1 [==============================] - 0s 114ms/step
1/1 [==============================] - 0s 135ms/step
1/1 [==============================] - 0s 124ms/step
1/1 [==============================] - 0s 95ms/step
1/1 [==============================] - 0s 90ms/step
1/1 [==============================] - 0s 95ms/step
1/1 [==============================] - 0s 99ms/step
1/1 [==============================] - 0s 104ms/step
1/1 [==============================] - 0s 128ms/step
1/1 [==============================] - 0s 130ms/step
1/1 [==============================] - 0s 94ms/step
1/1 [==============================] - 0s 129ms/step
1/1 [==============================] - 0s 93ms/step
1/1 [==============================] - 0s 112ms/step
1/1 [==============================] - 0s 129ms/step
1/1 [==============================] - 0s 126ms/step
1/1 [==============================] - 0s 142ms/step
1/1 [==============================] - 0s 131ms/step
1/1 [==============================] - 0s 196ms/step
1/1 [==============================] - 0s 113ms/step
1/1 [==============================] - 0s 183ms/step
1/1 [==============================] - 0s 131ms/step
1/1 [==============================] - 0s 165ms/step
1/1 [==============================] - 0s 126ms/step
1/1 [==============================] - 0s 148ms/step
1/1 [==============================] - 0s 133ms/step
1/1 [==============================] - 0s 169ms/step
1/1 [==============================] - 0s 121ms/step
1/1 [==============================] - 0s 115ms/step
1/1 [==============================] - 0s 173ms/step
1/1 [==============================] - 0s 133ms/step
1/1 [==============================] - 0s 101ms/step
1/1 [==============================] - 0s 149ms/step
1/1 [==============================] - 0s 179ms/step
1/1 [==============================] - 0s 178ms/step
```

```
1/1 [==============================] - 0s 131ms/step
1/1 [==============================] - 0s 198ms/step
1/1 [==============================] - 0s 163ms/step
1/1 [==============================] - 0s 152ms/step
1/1 [==============================] - 0s 161ms/step
1/1 [==============================] - 0s 152ms/step
1/1 [==============================] - 0s 168ms/step
1/1 [==============================] - 0s 98ms/step
1/1 [==============================] - 0s 123ms/step
1/1 [==============================] - 0s 150ms/step
1/1 [==============================] - 0s 151ms/step
1/1 [==============================] - 0s 165ms/step
1/1 [==============================] - 0s 233ms/step
1/1 [==============================] - 0s 175ms/step
1/1 [==============================] - 0s 152ms/step
1/1 [==============================] - 0s 160ms/step
1/1 [==============================] - 0s 160ms/step
1/1 [==============================] - 0s 137ms/step
1/1 [==============================] - 0s 118ms/step
1/1 [==============================] - 0s 127ms/step
1/1 [==============================] - 0s 149ms/step
1/1 [==============================] - 0s 142ms/step
1/1 [==============================] - 0s 196ms/step
1/1 [==============================] - 0s 123ms/step
1/1 [==============================] - 0s 130ms/step
1/1 [==============================] - 0s 108ms/step
1/1 [==============================] - 0s 150ms/step
1/1 [==============================] - 0s 136ms/step
1/1 [==============================] - 0s 124ms/step
1/1 [==============================] - 0s 118ms/step
1/1 [==============================] - 0s 135ms/step
1/1 [==============================] - 0s 113ms/step
1/1 [==============================] - 0s 152ms/step
1/1 [==============================] - 0s 135ms/step
1/1 [==============================] - 0s 126ms/step
1/1 [==============================] - 0s 154ms/step
1/1 [==============================] - 0s 134ms/step
1/1 [==============================] - 0s 143ms/step
1/1 [==============================] - 0s 133ms/step
1/1 [==============================] - 0s 142ms/step
1/1 [==============================] - 0s 127ms/step
1/1 [==============================] - 0s 124ms/step
1/1 [==============================] - 0s 134ms/step
1/1 [==============================] - 0s 167ms/step
1/1 [==============================] - 0s 182ms/step
1/1 [==============================] - 0s 137ms/step
1/1 [==============================] - 0s 126ms/step
1/1 [==============================] - 0s 144ms/step
```

```
1/1 [==============================] - 0s 133ms/step
1/1 [==============================] - 0s 134ms/step
1/1 [==============================] - 0s 108ms/step
1/1 [==============================] - 0s 133ms/step
1/1 [==============================] - 0s 129ms/step
1/1 [==============================] - 0s 121ms/step
1/1 [==============================] - 0s 166ms/step
1/1 [==============================] - 0s 145ms/step
1/1 [==============================] - 0s 177ms/step
1/1 [==============================] - 0s 128ms/step
1/1 [==============================] - 0s 114ms/step
1/1 [==============================] - 0s 121ms/step
1/1 [==============================] - 0s 121ms/step
1/1 [==============================] - 0s 113ms/step
1/1 [==============================] - 0s 158ms/step
1/1 [==============================] - 0s 96ms/step
1/1 [==============================] - 0s 129ms/step
1/1 [==============================] - 0s 160ms/step
1/1 [==============================] - 0s 202ms/step
1/1 [==============================] - 0s 127ms/step
1/1 [==============================] - 0s 119ms/step
1/1 [==============================] - 0s 120ms/step
1/1 [==============================] - 0s 134ms/step
1/1 [==============================] - 0s 132ms/step
1/1 [==============================] - 0s 134ms/step
1/1 [==============================] - 0s 156ms/step
1/1 [==============================] - 0s 147ms/step
1/1 [==============================] - 0s 100ms/step
1/1 [==============================] - 0s 121ms/step
1/1 [==============================] - 0s 122ms/step
1/1 [==============================] - 0s 125ms/step
1/1 [==============================] - 0s 132ms/step
1/1 [==============================] - 0s 116ms/step
1/1 [==============================] - 0s 137ms/step
1/1 [==============================] - 0s 124ms/step
1/1 [==============================] - 0s 139ms/step
1/1 [==============================] - 0s 120ms/step
1/1 [==============================] - 0s 153ms/step
1/1 [==============================] - 0s 122ms/step
1/1 [==============================] - 0s 136ms/step
1/1 [==============================] - 0s 136ms/step
1/1 [==============================] - 0s 138ms/step
1/1 [==============================] - 0s 123ms/step
1/1 [==============================] - 0s 130ms/step
1/1 [==============================] - 0s 131ms/step
1/1 [==============================] - 0s 122ms/step
1/1 [==============================] - 0s 103ms/step
1/1 [==============================] - 0s 120ms/step
```

```
1/1 [==============================] - 0s 122ms/step
```

[11]:
```
(150, 1, 1000)
```

1.1.3 Step 3: Compute distances between the original image and each of the perturbed images and compute weights (importance) of each perturbed image

The distance between each randomly generated pertubation and the image being explained is computed using the cosine distance. For the shape of the `distances` array it can be noted that, as expected, there is a distance for every generated perturbation.

[17]:
```
original_image = np.ones(num_superpixels)[np.newaxis,:] #Perturbation with all␣
↪superpixels enabled
distances = sklearn.metrics.pairwise_distances(perturbations,original_image,␣
↪metric='cosine').ravel()
distances
```

[17]:
```
array([0.31899478, 0.27768488, 0.29803588, 0.24819059, 0.29803588,
       0.36297794, 0.28778769, 0.32971994, 0.32971994, 0.35170234,
       0.32971994, 0.28778769, 0.30843593, 0.28778769, 0.20145059,
       0.23861301, 0.34061953, 0.30843593, 0.38615019, 0.30843593,
       0.26772144, 0.37445676, 0.28778769, 0.25789176, 0.26772144,
       0.30843593, 0.27768488, 0.35170234, 0.30843593, 0.26772144,
       0.26772144, 0.24819059, 0.28778769, 0.32971994, 0.32971994,
       0.34061953, 0.31899478, 0.31899478, 0.28778769, 0.34061953,
       0.25789176, 0.34061953, 0.32971994, 0.23861301, 0.29803588,
       0.32971994, 0.35170234, 0.36297794, 0.35170234, 0.25789176,
       0.28778769, 0.26772144, 0.35170234, 0.35170234, 0.23861301,
       0.41023218, 0.30843593, 0.28778769, 0.32971994, 0.32971994,
       0.39807073, 0.30843593, 0.28778769, 0.32971994, 0.26772144,
       0.28778769, 0.24819059, 0.30843593, 0.28778769, 0.27768488,
       0.35170234, 0.14874347, 0.27768488, 0.23861301, 0.27768488,
       0.30843593, 0.35170234, 0.29803588, 0.26772144, 0.29803588,
       0.30843593, 0.34061953, 0.31899478, 0.32971994, 0.27768488,
       0.31899478, 0.31899478, 0.34061953, 0.35170234, 0.32971994,
       0.28778769, 0.28778769, 0.23861301, 0.31899478, 0.32971994,
       0.35170234, 0.21057717, 0.24819059, 0.29803588, 0.34061953,
       0.24819059, 0.29803588, 0.30843593, 0.23861301, 0.28778769,
       0.30843593, 0.24819059, 0.31899478, 0.34061953, 0.30843593,
       0.30843593, 0.27768488, 0.29803588, 0.27768488, 0.29803588,
       0.26772144, 0.28778769, 0.22915443, 0.34061953, 0.28778769,
       0.18350342, 0.2198105 , 0.25789176, 0.29803588, 0.24819059,
       0.29803588, 0.23861301, 0.31899478, 0.27768488, 0.29803588,
       0.35170234, 0.29803588, 0.26772144, 0.34061953, 0.34061953,
       0.28778769, 0.30843593, 0.29803588, 0.31899478, 0.27768488,
       0.30843593, 0.30843593, 0.30843593, 0.23861301, 0.26772144,
       0.30843593, 0.27768488, 0.25789176, 0.28778769, 0.35170234])
```

Use kernel function to compute weights The distances are then mapped to a value between zero and one (weight) using a kernel function. An example of a kernel function with different kernel widths is shown in the plot below. Here the x axis represents distances and the y axis the weights. Depeding on how we set the kernel width, it defines how wide we want the "locality" around our instance to be. This kernel width can be set based on expected distance values. For the case of cosine distances, we expect them to be somehow stable (between 0 and 1); therefore, no fine tunning of the kernel width might be required.

[18]:
```
kernel_width = 0.25
weights = np.sqrt(np.exp(-(distances**2)/kernel_width**2)) #Kernel function
weights
```

[18]:
```
array([0.44305501, 0.53963022, 0.49134778, 0.61092044, 0.49134778,
       0.34853247, 0.51552266, 0.41906808, 0.41906808, 0.3717416 ,
       0.41906808, 0.51552266, 0.46717014, 0.51552266, 0.72277398,
       0.63413757, 0.3952751 , 0.46717014, 0.3033416 , 0.46717014,
       0.56360695, 0.32571161, 0.51552266, 0.58739059, 0.56360695,
       0.46717014, 0.53963022, 0.3717416 , 0.46717014, 0.56360695,
       0.56360695, 0.61092044, 0.51552266, 0.41906808, 0.41906808,
       0.3952751 , 0.44305501, 0.44305501, 0.51552266, 0.3952751 ,
       0.58739059, 0.3952751 , 0.41906808, 0.63413757, 0.49134778,
       0.41906808, 0.3717416 , 0.34853247, 0.3717416 , 0.58739059,
       0.51552266, 0.56360695, 0.3717416 , 0.3717416 , 0.63413757,
       0.26019511, 0.46717014, 0.51552266, 0.41906808, 0.41906808,
       0.28148321, 0.46717014, 0.51552266, 0.41906808, 0.56360695,
       0.51552266, 0.61092044, 0.46717014, 0.51552266, 0.53963022,
       0.3717416 , 0.83778233, 0.53963022, 0.63413757, 0.53963022,
       0.46717014, 0.3717416 , 0.49134778, 0.56360695, 0.49134778,
       0.46717014, 0.3952751 , 0.44305501, 0.41906808, 0.53963022,
       0.44305501, 0.44305501, 0.3952751 , 0.3717416 , 0.41906808,
       0.51552266, 0.51552266, 0.63413757, 0.44305501, 0.41906808,
       0.3717416 , 0.7013544 , 0.61092044, 0.49134778, 0.3952751 ,
       0.61092044, 0.49134778, 0.46717014, 0.63413757, 0.51552266,
       0.46717014, 0.61092044, 0.44305501, 0.3952751 , 0.46717014,
       0.46717014, 0.53963022, 0.49134778, 0.53963022, 0.49134778,
       0.56360695, 0.51552266, 0.65698505, 0.3952751 , 0.51552266,
       0.7638468 , 0.67940813, 0.58739059, 0.49134778, 0.61092044,
       0.49134778, 0.63413757, 0.44305501, 0.53963022, 0.49134778,
       0.3717416 , 0.49134778, 0.56360695, 0.3952751 , 0.3952751 ,
       0.51552266, 0.46717014, 0.49134778, 0.44305501, 0.53963022,
       0.46717014, 0.46717014, 0.46717014, 0.63413757, 0.56360695,
       0.46717014, 0.53963022, 0.58739059, 0.51552266, 0.3717416 ])
```

1.1.4 Step 4: Use `perturbations`, `predictions` and `weights` to fit an explainable (linear) model

A weighed linear regression model is fitted using data from the previous steps (perturbations, predictions and weights). Given that the class that we want to explain is labrador, when fitting the

linear model we take from the predictions vector only the column corresponding to the top predicted class. Each coefficients in the linear model corresponds to one superpixel in the segmented image. These coefficients represent how important is each superpixel for the prediction of labrador.

[25]:
```
class_to_explain = top_pred_classes[0]
simpler_model = LinearRegression()
simpler_model.fit(X=perturbations, y=predictions[:,:,class_to_explain],␣
 ↪sample_weight=weights)
# perturbations: superpixels being turned on or off (bool array), predictions:␣
 ↪prediction probability that an image with blacked superpixels belongs to␣
 ↪class labrador
# weights: distance from the class Labrador to prediction (the further away a␣
 ↪prediction is the less it counts)
coeff = simpler_model.coef_[0]
coeff
```

[25]:
```
array([ 0.02147114, -0.09216851,  0.02872054, -0.0319038 ,  0.00916225,
        0.05393283,  0.08358517, -0.00498543,  0.27438518,  0.09585625,
        0.00296036,  0.01264599, -0.04494346,  0.07178767,  0.01563819,
        0.03242433, -0.01918968, -0.01105364,  0.01820046,  0.0300082 ,
       -0.03831444, -0.05082932, -0.02133498, -0.01711476,  0.07428526,
        0.47752497,  0.08612496,  0.03121331, -0.0520179 ,  0.11176997,
        0.05881402,  0.03161452,  0.01681638, -0.09078893,  0.05338044,
       -0.03416311, -0.04749763,  0.07826562, -0.06273997, -0.06259037,
       -0.01732085, -0.00892435,  0.01199878,  0.00661503, -0.06392813,
        0.01328099, -0.05739886, -0.01436495,  0.04267476, -0.03402085,
        0.09734626, -0.00247384,  0.05974761,  0.02024142, -0.11034672,
       -0.02408094, -0.01114504,  0.01829057,  0.02682509,  0.0442639 ,
       -0.0440699 , -0.02364172,  0.06476514, -0.11566973,  0.05263674,
        0.01568603,  0.04484574,  0.00198514,  0.06745345])
```

[32]:
```
perturbations.shape, predictions[:, :, class_to_explain].shape, coeff.shape
```

[32]:
```
((150, 69), (150, 1), (69,))
```

Compute top features (superpixels) Now, sort the coefficients to figure out which are the supperpixels, that have larger coefficients (magnitude) for the prediction of labradors. The identifiers of these top features or superpixels are shown below. Even though here the magnitude of the coefficients were used to determine the most important features, other alternatives such as forward or backward elimination can be used for feature importance selection.

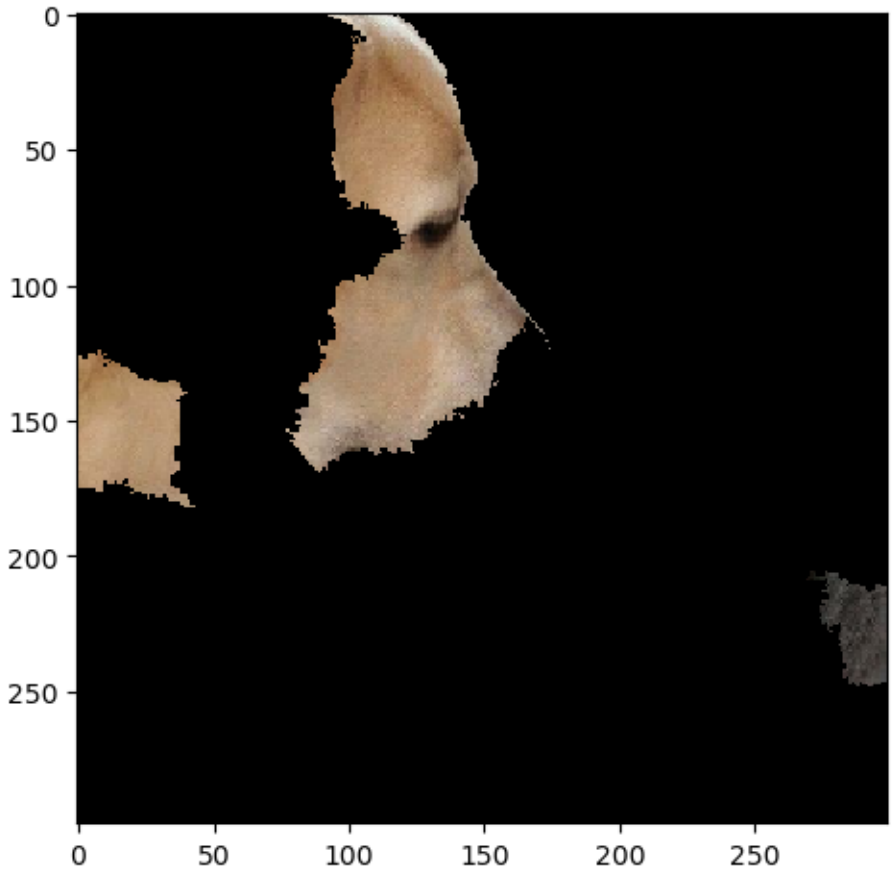
[15]:
```
num_top_features = 4
top_features = np.argsort(coeff)[-num_top_features:]
top_features
```

[15]:
```
array([50, 29,  8, 25])
```

Show LIME explanation (image with top features) Let's show the most important superpixels defined in the previous step in an image after covering up less relevant superpixels.

[16]:
```
mask = np.zeros(num_superpixels)
mask[top_features]= True #Activate top superpixels
skimage.io.imshow(perturb_image(Xi/2+0.5,mask,superpixels))
```

[16]: <matplotlib.image.AxesImage at 0x7f91f0eb9b10>

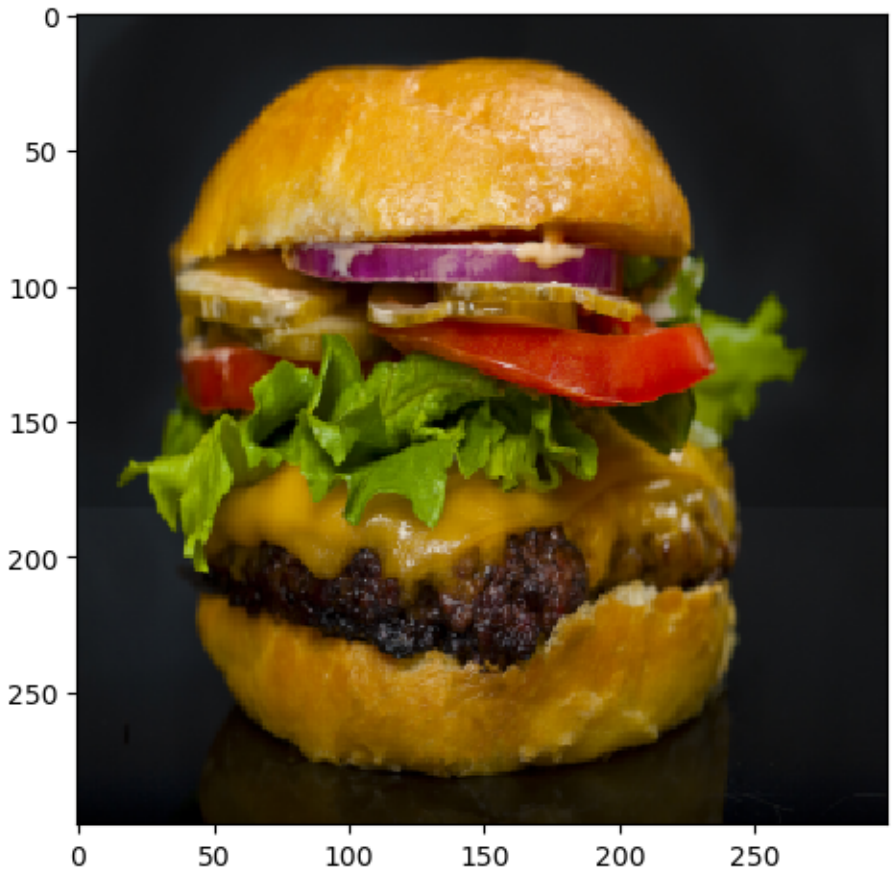


This is the final step where we obtain the area of the image that produced the prediction of labrador.

1.2 Selfmade

[40]:
```
Xi = skimage.io.imread("./data/chesseburger.jpg")
Xi = skimage.transform.resize(Xi, (299,299))
Xi = (Xi - 0.5)*2 #Inception pre-processing
skimage.io.imshow(Xi/2+0.5) # Show image before inception preprocessing
```

[40]: <matplotlib.image.AxesImage at 0x7f916bf4ac90>



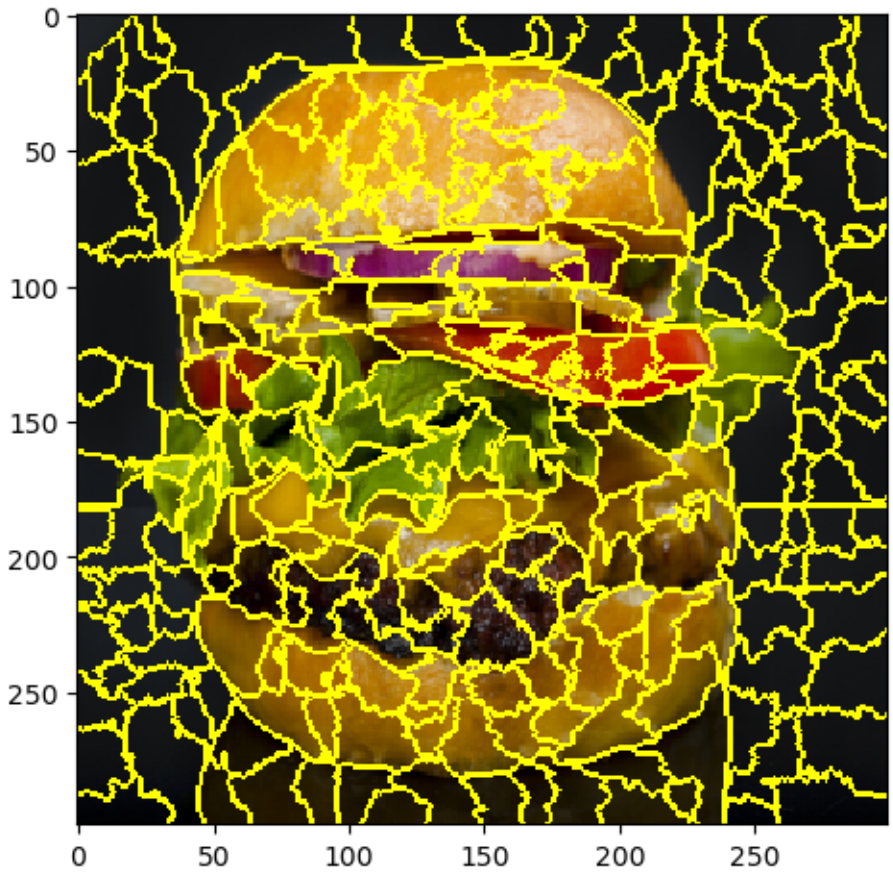
[41]:
```
preds = inceptionV3_model.predict(Xi[np.newaxis,:,:,:])
decode_predictions(preds)[0] #Top 5 classes
```

```
1/1 [==============================] - 0s 151ms/step
```

[41]:
```
[('n07697313', 'cheeseburger', 0.93166506),
 ('n04004767', 'printer', 0.033531126),
 ('n03924679', 'photocopier', 0.000737663),
 ('n07871810', 'meat_loaf', 0.0007184044),
 ('n03770679', 'minivan', 0.000182352)]
```

[74]:
```
superpixels = skimage.segmentation.quickshift(Xi, kernel_size=2,max_dist=200,␣
↪ratio=0.2)
num_superpixels = np.unique(superpixels).shape[0]
skimage.io.imshow(skimage.segmentation.mark_boundaries(Xi/2+0.5, superpixels))
```

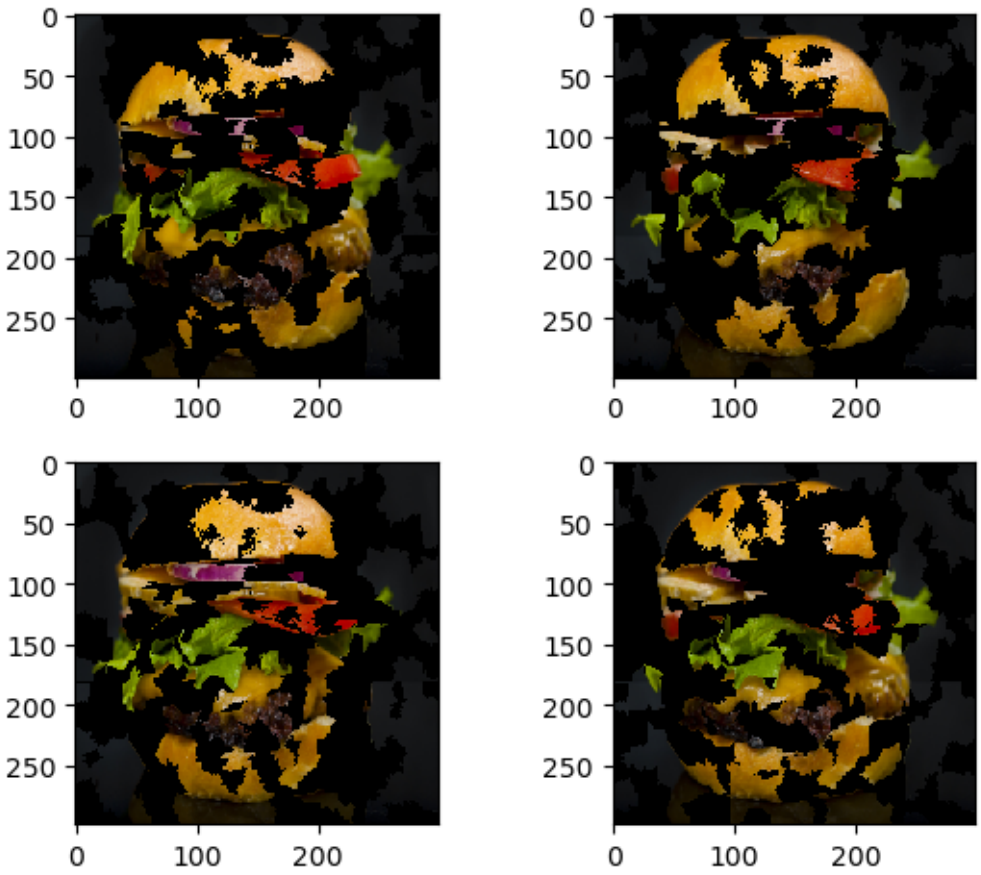
[74]:
```
<matplotlib.image.AxesImage at 0x7f916a68ee90>
```



[75]:
```
num_perturb = 300
perturbations = np.random.binomial(1, 0.5, size=(num_perturb, num_superpixels))
perturbations.shape
```

[75]: (300, 246)

[76]:
```
fig, axs = plt.subplots(2, 2)
for ax in axs.flatten():
    skimage.io.imshow(perturb_image(Xi/2+0.5,perturbations[np.random.
 ↪choice(num_perturb)],superpixels), ax=ax)
plt.show()
```



[80]:
```
predictions = []
for ptbt in perturbations:
    predictions.append(inceptionV3_model.predict(perturb_image(Xi/2+0.
 ↪5,ptbt,superpixels)[np.newaxis,:,:,:]))
```

```
1/1 [==============================] - 0s 97ms/step
1/1 [==============================] - 0s 170ms/step
1/1 [==============================] - 0s 109ms/step
1/1 [==============================] - 0s 120ms/step
1/1 [==============================] - 0s 124ms/step
1/1 [==============================] - 0s 135ms/step
1/1 [==============================] - 0s 136ms/step
1/1 [==============================] - 0s 125ms/step
1/1 [==============================] - 0s 116ms/step
1/1 [==============================] - 0s 111ms/step
1/1 [==============================] - 0s 121ms/step
1/1 [==============================] - 0s 112ms/step
1/1 [==============================] - 0s 128ms/step
1/1 [==============================] - 0s 136ms/step
```

```
1/1 [==============================] - 0s 120ms/step
1/1 [==============================] - 0s 124ms/step
1/1 [==============================] - 0s 165ms/step
1/1 [==============================] - 0s 189ms/step
1/1 [==============================] - 0s 145ms/step
1/1 [==============================] - 0s 122ms/step
1/1 [==============================] - 0s 122ms/step
1/1 [==============================] - 0s 119ms/step
1/1 [==============================] - 0s 111ms/step
1/1 [==============================] - 0s 149ms/step
1/1 [==============================] - 0s 146ms/step
1/1 [==============================] - 0s 109ms/step
1/1 [==============================] - 0s 129ms/step
1/1 [==============================] - 0s 154ms/step
1/1 [==============================] - 0s 141ms/step
1/1 [==============================] - 0s 124ms/step
1/1 [==============================] - 0s 133ms/step
1/1 [==============================] - 0s 126ms/step
1/1 [==============================] - 0s 137ms/step
1/1 [==============================] - 0s 131ms/step
1/1 [==============================] - 0s 115ms/step
1/1 [==============================] - 0s 84ms/step
1/1 [==============================] - 0s 113ms/step
1/1 [==============================] - 0s 116ms/step
1/1 [==============================] - 0s 112ms/step
1/1 [==============================] - 0s 103ms/step
1/1 [==============================] - 0s 115ms/step
1/1 [==============================] - 0s 128ms/step
1/1 [==============================] - 0s 131ms/step
1/1 [==============================] - 0s 138ms/step
1/1 [==============================] - 0s 125ms/step
1/1 [==============================] - 0s 124ms/step
1/1 [==============================] - 0s 138ms/step
1/1 [==============================] - 0s 130ms/step
1/1 [==============================] - 0s 121ms/step
1/1 [==============================] - 0s 121ms/step
1/1 [==============================] - 0s 134ms/step
1/1 [==============================] - 0s 121ms/step
1/1 [==============================] - 0s 108ms/step
1/1 [==============================] - 0s 109ms/step
1/1 [==============================] - 0s 103ms/step
1/1 [==============================] - 0s 111ms/step
1/1 [==============================] - 0s 101ms/step
1/1 [==============================] - 0s 127ms/step
1/1 [==============================] - 0s 111ms/step
1/1 [==============================] - 0s 118ms/step
1/1 [==============================] - 0s 116ms/step
1/1 [==============================] - 0s 116ms/step
```

```
1/1 [==============================] - 0s 131ms/step
1/1 [==============================] - 0s 118ms/step
1/1 [==============================] - 0s 84ms/step
1/1 [==============================] - 0s 79ms/step
1/1 [==============================] - 0s 77ms/step
1/1 [==============================] - 0s 125ms/step
1/1 [==============================] - 0s 115ms/step
1/1 [==============================] - 0s 134ms/step
1/1 [==============================] - 0s 119ms/step
1/1 [==============================] - 0s 122ms/step
1/1 [==============================] - 0s 105ms/step
1/1 [==============================] - 0s 122ms/step
1/1 [==============================] - 0s 113ms/step
1/1 [==============================] - 0s 118ms/step
1/1 [==============================] - 0s 130ms/step
1/1 [==============================] - 0s 119ms/step
1/1 [==============================] - 0s 110ms/step
1/1 [==============================] - 0s 117ms/step
1/1 [==============================] - 0s 134ms/step
1/1 [==============================] - 0s 117ms/step
1/1 [==============================] - 0s 126ms/step
1/1 [==============================] - 0s 132ms/step
1/1 [==============================] - 0s 131ms/step
1/1 [==============================] - 0s 103ms/step
1/1 [==============================] - 0s 127ms/step
1/1 [==============================] - 0s 121ms/step
1/1 [==============================] - 0s 110ms/step
1/1 [==============================] - 0s 110ms/step
1/1 [==============================] - 0s 123ms/step
1/1 [==============================] - 0s 116ms/step
1/1 [==============================] - 0s 116ms/step
1/1 [==============================] - 0s 120ms/step
1/1 [==============================] - 0s 121ms/step
1/1 [==============================] - 0s 94ms/step
1/1 [==============================] - 0s 131ms/step
1/1 [==============================] - 0s 132ms/step
1/1 [==============================] - 0s 119ms/step
1/1 [==============================] - 0s 129ms/step
1/1 [==============================] - 0s 144ms/step
1/1 [==============================] - 0s 139ms/step
1/1 [==============================] - 0s 138ms/step
1/1 [==============================] - 0s 123ms/step
1/1 [==============================] - 0s 119ms/step
1/1 [==============================] - 0s 119ms/step
1/1 [==============================] - 0s 133ms/step
1/1 [==============================] - 0s 118ms/step
1/1 [==============================] - 0s 128ms/step
1/1 [==============================] - 0s 135ms/step
```

```
1/1 [==============================] - 0s 126ms/step
1/1 [==============================] - 0s 118ms/step
1/1 [==============================] - 0s 120ms/step
1/1 [==============================] - 0s 125ms/step
1/1 [==============================] - 0s 133ms/step
1/1 [==============================] - 0s 138ms/step
1/1 [==============================] - 0s 133ms/step
1/1 [==============================] - 0s 117ms/step
1/1 [==============================] - 0s 149ms/step
1/1 [==============================] - 0s 165ms/step
1/1 [==============================] - 0s 141ms/step
1/1 [==============================] - 0s 127ms/step
1/1 [==============================] - 0s 127ms/step
1/1 [==============================] - 0s 119ms/step
1/1 [==============================] - 0s 120ms/step
1/1 [==============================] - 0s 117ms/step
1/1 [==============================] - 0s 147ms/step
1/1 [==============================] - 0s 122ms/step
1/1 [==============================] - 0s 122ms/step
1/1 [==============================] - 0s 119ms/step
1/1 [==============================] - 0s 125ms/step
1/1 [==============================] - 0s 161ms/step
1/1 [==============================] - 0s 153ms/step
1/1 [==============================] - 0s 145ms/step
1/1 [==============================] - 0s 115ms/step
1/1 [==============================] - 0s 155ms/step
1/1 [==============================] - 0s 138ms/step
1/1 [==============================] - 0s 130ms/step
1/1 [==============================] - 0s 162ms/step
1/1 [==============================] - 0s 96ms/step
1/1 [==============================] - 0s 121ms/step
1/1 [==============================] - 0s 119ms/step
1/1 [==============================] - 0s 168ms/step
1/1 [==============================] - 0s 228ms/step
1/1 [==============================] - 0s 228ms/step
1/1 [==============================] - 0s 179ms/step
1/1 [==============================] - 0s 210ms/step
1/1 [==============================] - 0s 154ms/step
1/1 [==============================] - 0s 156ms/step
1/1 [==============================] - 0s 218ms/step
1/1 [==============================] - 0s 217ms/step
1/1 [==============================] - 0s 212ms/step
1/1 [==============================] - 0s 185ms/step
1/1 [==============================] - 0s 124ms/step
1/1 [==============================] - 0s 149ms/step
1/1 [==============================] - 0s 167ms/step
1/1 [==============================] - 0s 184ms/step
1/1 [==============================] - 0s 224ms/step
```

```
1/1 [==============================] - 0s 222ms/step
1/1 [==============================] - 0s 226ms/step
1/1 [==============================] - 0s 226ms/step
1/1 [==============================] - 0s 247ms/step
1/1 [==============================] - 0s 183ms/step
1/1 [==============================] - 0s 197ms/step
1/1 [==============================] - 0s 219ms/step
1/1 [==============================] - 0s 180ms/step
1/1 [==============================] - 0s 195ms/step
1/1 [==============================] - 0s 230ms/step
1/1 [==============================] - 0s 169ms/step
1/1 [==============================] - 0s 136ms/step
1/1 [==============================] - 0s 138ms/step
1/1 [==============================] - 0s 131ms/step
1/1 [==============================] - 0s 151ms/step
1/1 [==============================] - 0s 142ms/step
1/1 [==============================] - 0s 195ms/step
1/1 [==============================] - 0s 189ms/step
1/1 [==============================] - 0s 176ms/step
1/1 [==============================] - 0s 157ms/step
1/1 [==============================] - 0s 154ms/step
1/1 [==============================] - 0s 165ms/step
1/1 [==============================] - 0s 145ms/step
1/1 [==============================] - 0s 179ms/step
1/1 [==============================] - 0s 224ms/step
1/1 [==============================] - 0s 169ms/step
1/1 [==============================] - 0s 168ms/step
1/1 [==============================] - 0s 120ms/step
1/1 [==============================] - 0s 161ms/step
1/1 [==============================] - 0s 145ms/step
1/1 [==============================] - 0s 146ms/step
1/1 [==============================] - 0s 160ms/step
1/1 [==============================] - 0s 158ms/step
1/1 [==============================] - 0s 174ms/step
1/1 [==============================] - 0s 124ms/step
1/1 [==============================] - 0s 119ms/step
1/1 [==============================] - 0s 151ms/step
1/1 [==============================] - 0s 151ms/step
1/1 [==============================] - 0s 178ms/step
1/1 [==============================] - 0s 131ms/step
1/1 [==============================] - 0s 190ms/step
1/1 [==============================] - 0s 208ms/step
1/1 [==============================] - 0s 189ms/step
1/1 [==============================] - 0s 220ms/step
1/1 [==============================] - 0s 233ms/step
1/1 [==============================] - 0s 157ms/step
1/1 [==============================] - 0s 209ms/step
1/1 [==============================] - 0s 162ms/step
```

```
1/1 [==============================] - 0s 172ms/step
1/1 [==============================] - 0s 156ms/step
1/1 [==============================] - 0s 170ms/step
1/1 [==============================] - 0s 175ms/step
1/1 [==============================] - 0s 181ms/step
1/1 [==============================] - 0s 204ms/step
1/1 [==============================] - 0s 240ms/step
1/1 [==============================] - 0s 173ms/step
1/1 [==============================] - 0s 151ms/step
1/1 [==============================] - 0s 323ms/step
1/1 [==============================] - 0s 302ms/step
1/1 [==============================] - 0s 196ms/step
1/1 [==============================] - 0s 223ms/step
1/1 [==============================] - 0s 234ms/step
1/1 [==============================] - 0s 203ms/step
1/1 [==============================] - 0s 189ms/step
1/1 [==============================] - 0s 179ms/step
1/1 [==============================] - 0s 178ms/step
1/1 [==============================] - 0s 197ms/step
1/1 [==============================] - 0s 175ms/step
1/1 [==============================] - 0s 133ms/step
1/1 [==============================] - 0s 138ms/step
1/1 [==============================] - 0s 227ms/step
1/1 [==============================] - 0s 200ms/step
1/1 [==============================] - 0s 162ms/step
1/1 [==============================] - 0s 143ms/step
1/1 [==============================] - 0s 125ms/step
1/1 [==============================] - 0s 145ms/step
1/1 [==============================] - 0s 188ms/step
1/1 [==============================] - 0s 293ms/step
1/1 [==============================] - 0s 189ms/step
1/1 [==============================] - 0s 212ms/step
1/1 [==============================] - 0s 219ms/step
1/1 [==============================] - 0s 207ms/step
1/1 [==============================] - 0s 190ms/step
1/1 [==============================] - 0s 192ms/step
1/1 [==============================] - 0s 188ms/step
1/1 [==============================] - 0s 224ms/step
1/1 [==============================] - 0s 166ms/step
1/1 [==============================] - 0s 168ms/step
1/1 [==============================] - 0s 158ms/step
1/1 [==============================] - 0s 176ms/step
1/1 [==============================] - 0s 190ms/step
1/1 [==============================] - 0s 153ms/step
1/1 [==============================] - 0s 149ms/step
1/1 [==============================] - 0s 217ms/step
1/1 [==============================] - 0s 143ms/step
1/1 [==============================] - 0s 122ms/step
```

```
1/1 [==============================] - 0s 178ms/step
1/1 [==============================] - 0s 208ms/step
1/1 [==============================] - 0s 165ms/step
1/1 [==============================] - 0s 173ms/step
1/1 [==============================] - 0s 175ms/step
1/1 [==============================] - 0s 173ms/step
1/1 [==============================] - 0s 175ms/step
1/1 [==============================] - 0s 170ms/step
1/1 [==============================] - 0s 171ms/step
1/1 [==============================] - 0s 160ms/step
1/1 [==============================] - 0s 164ms/step
1/1 [==============================] - 0s 197ms/step
1/1 [==============================] - 0s 175ms/step
1/1 [==============================] - 0s 171ms/step
1/1 [==============================] - 0s 174ms/step
1/1 [==============================] - 0s 164ms/step
1/1 [==============================] - 0s 141ms/step
1/1 [==============================] - 0s 159ms/step
1/1 [==============================] - 0s 175ms/step
1/1 [==============================] - 0s 146ms/step
1/1 [==============================] - 0s 203ms/step
1/1 [==============================] - 0s 165ms/step
1/1 [==============================] - 0s 193ms/step
1/1 [==============================] - 0s 204ms/step
1/1 [==============================] - 0s 177ms/step
1/1 [==============================] - 0s 205ms/step
1/1 [==============================] - 0s 198ms/step
1/1 [==============================] - 0s 180ms/step
1/1 [==============================] - 0s 191ms/step
1/1 [==============================] - 0s 192ms/step
1/1 [==============================] - 0s 216ms/step
1/1 [==============================] - 0s 204ms/step
1/1 [==============================] - 0s 193ms/step
1/1 [==============================] - 0s 197ms/step
1/1 [==============================] - 0s 183ms/step
1/1 [==============================] - 0s 251ms/step
1/1 [==============================] - 0s 227ms/step
1/1 [==============================] - 0s 221ms/step
1/1 [==============================] - 0s 169ms/step
1/1 [==============================] - 0s 237ms/step
1/1 [==============================] - 0s 172ms/step
1/1 [==============================] - 0s 194ms/step
1/1 [==============================] - 0s 183ms/step
1/1 [==============================] - 0s 174ms/step
1/1 [==============================] - 0s 210ms/step
1/1 [==============================] - 0s 211ms/step
```

[82]:
```
predictions = np.array(predictions)
predictions.shape
```

[82]:
```
(300, 1, 1000)
```

[84]:
```
original_image = np.ones(num_superpixels)[np.newaxis,:] #Perturbation with all␣
 ↪superpixels enabled
distances = sklearn.metrics.pairwise_distances(perturbations,original_image,␣
 ↪metric='cosine').ravel()
weights = np.sqrt(np.exp(-(distances**2)/0.45**2)) #Kernel function
weights
```

[84]:
```
array([0.84414691, 0.80573139, 0.80231839, 0.83806816, 0.83186045,
       0.7988726 , 0.78833808, 0.79539398, 0.76637854, 0.82230612,
       0.80231839, 0.8124592 , 0.8287081 , 0.77750704, 0.81905639,
       0.83806816, 0.85009717, 0.8124592 , 0.78833808, 0.7988726 ,
       0.8157741 , 0.83806816, 0.80231839, 0.79539398, 0.84414691,
       0.80911165, 0.82552334, 0.77750704, 0.82230612, 0.8287081 ,
       0.80573139, 0.7988726 , 0.7811504 , 0.7988726 , 0.79539398,
       0.80573139, 0.86161419, 0.83186045, 0.7811504 , 0.78833808,
       0.77383064, 0.84713807, 0.7811504 , 0.77383064, 0.7988726 ,
       0.80231839, 0.83498045, 0.8157741 , 0.75495176, 0.8287081 ,
       0.80573139, 0.78476073, 0.75495176, 0.84414691, 0.85302427,
       0.82230612, 0.8124592 , 0.81905639, 0.7811504 , 0.85878271,
       0.83498045, 0.81905639, 0.80573139, 0.82230612, 0.73925083,
       0.80911165, 0.8157741 , 0.78833808, 0.80573139, 0.79188249,
       0.73524238, 0.83806816, 0.77012115, 0.78833808, 0.83186045,
       0.8124592 , 0.80911165, 0.78833808, 0.8287081 , 0.85878271,
       0.77383064, 0.84112362, 0.79188249, 0.86441393, 0.80231839,
       0.83186045, 0.77383064, 0.76637854, 0.7811504 , 0.83806816,
       0.79539398, 0.82552334, 0.78833808, 0.82230612, 0.83498045,
       0.75495176, 0.75495176, 0.80231839, 0.81905639, 0.78833808,
       0.86441393, 0.78476073, 0.82552334, 0.82552334, 0.8157741 ,
       0.80911165, 0.8287081 , 0.85878271, 0.7988726 , 0.7988726 ,
       0.80573139, 0.8287081 , 0.83186045, 0.83806816, 0.80573139,
       0.8157741 , 0.74716783, 0.83498045, 0.78833808, 0.83806816,
       0.80231839, 0.84713807, 0.78476073, 0.77012115, 0.84112362,
       0.83498045, 0.82552334, 0.82230612, 0.82552334, 0.80573139,
       0.82230612, 0.78833808, 0.78833808, 0.8287081 , 0.80573139,
       0.8157741 , 0.82552334, 0.77012115, 0.83186045, 0.77750704,
       0.80573139, 0.73524238, 0.7811504 , 0.80231839, 0.8287081 ,
       0.74322598, 0.82552334, 0.78833808, 0.78476073, 0.8157741 ,
       0.80573139, 0.8124592 , 0.78476073, 0.81905639, 0.82552334,
       0.83498045, 0.7988726 , 0.81905639, 0.8124592 , 0.8157741 ,
       0.77750704, 0.84414691, 0.80573139, 0.77012115, 0.84414691,
       0.80573139, 0.84713807, 0.84414691, 0.87529685, 0.82230612,
       0.7811504 , 0.77383064, 0.80911165, 0.84713807, 0.80911165,
```

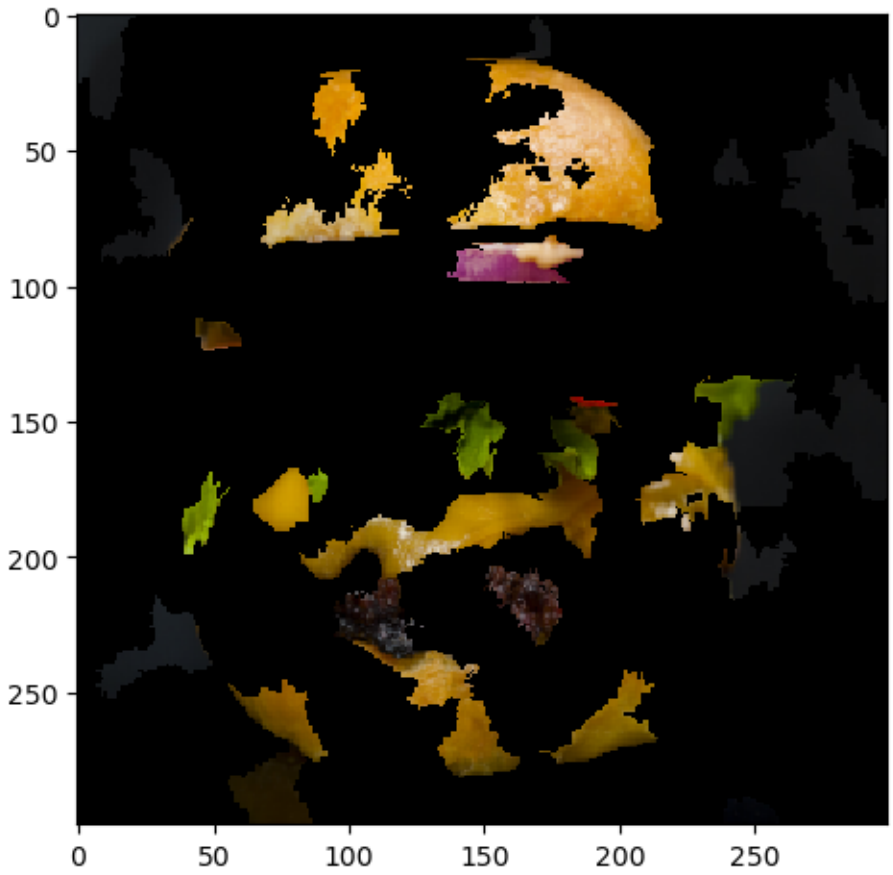
```
0.80911165, 0.7988726 , 0.7811504 , 0.84414691, 0.80911165,
0.78476073, 0.8157741 , 0.84414691, 0.8157741 , 0.78833808,
0.78476073, 0.84713807, 0.74322598, 0.78476073, 0.79188249,
0.77012115, 0.84112362, 0.80911165, 0.80231839, 0.79188249,
0.8124592 , 0.81905639, 0.8287081 , 0.81905639, 0.78476073,
0.8124592 , 0.80573139, 0.82552334, 0.84414691, 0.8157741 ,
0.83186045, 0.80911165, 0.7988726 , 0.80573139, 0.7626028 ,
0.7811504 , 0.80911165, 0.8124592 , 0.79188249, 0.84713807,
0.83498045, 0.80573139, 0.75107642, 0.84112362, 0.84112362,
0.8124592 , 0.8287081 , 0.82552334, 0.80231839, 0.78833808,
0.8124592 , 0.84112362, 0.8124592 , 0.7988726 , 0.83498045,
0.8124592 , 0.85009717, 0.83498045, 0.8287081 , 0.82552334,
0.81905639, 0.85878271, 0.74716783, 0.77750704, 0.83806816,
0.83498045, 0.8287081 , 0.78833808, 0.86991846, 0.80573139,
0.83186045, 0.8124592 , 0.80231839, 0.76637854, 0.84414691,
0.81905639, 0.81905639, 0.86161419, 0.84713807, 0.85302427,
0.77750704, 0.7988726 , 0.80911165, 0.7988726 , 0.82230612,
0.8287081 , 0.8287081 , 0.8287081 , 0.7811504 , 0.8157741 ,
0.81905639, 0.77012115, 0.81905639, 0.7988726 , 0.84713807,
0.81905639, 0.77383064, 0.83498045, 0.80573139, 0.80573139,
0.84112362, 0.79188249, 0.85302427, 0.7811504 , 0.82230612,
0.8157741 , 0.80573139, 0.76637854, 0.7988726 , 0.75495176,
0.80573139, 0.84414691, 0.85302427, 0.7988726 , 0.80231839,
0.8157741 , 0.84713807, 0.8157741 , 0.86991846, 0.80573139,
0.77383064, 0.83806816, 0.80573139, 0.84112362, 0.8157741 ])
```

[100]:
```
simple_model = LinearRegression()
simple_model.fit(X=perturbations, y=predictions[:, :, np.argmax(preds)],␣
↪sample_weight=weights)
```

[100]:
```
LinearRegression()
```

[147]:
```
mask = np.zeros_like(perturbations)
mask[np.argsort(simple_model.coef_)[:, -50:]] = 1
skimage.io.imshow(perturb_image(Xi/2+0.5, mask, superpixels))
```

[147]:
```
<matplotlib.image.AxesImage at 0x7f916a4b8610>
```



LIME is here used to find the most sigificant superpixels. The complex model is replaced with a much simpler linear model.

[ ]: