Neural Network Interpretation

Integrated Gradients [1]

> "Attributing the predictions of Deep Networks to input Features"

> For linear models, ML practitioners regularlyinspect the products of the model coefficients and the fea-ture values in order to debug predictions. Gradients (of theoutput with respect to the input) is a natural analog of themodel coefficients for a deep network.

Axioms

- Sensitivity : If two input x_1 and x_2 differ in **one feature** and have different predictions $F(x_1)$ and $F(x_2)$ then **the differing feature should be given a non-zero attribution**.

It is apparently sometimes not assured in cases like $Relu$ function and cause problems. Other methods like DeepLift handle it wif specific backpropagation logic.

> "Unfortunately, Deconvolution networks (DeConvNets),and Guided back-propagation violate Sensitivity(a). Thisis because these methods back-propogate through a ReLUnode only if the ReLU is turned on at the input."

- Implementation Invariance : If two networks F_a and F_b are **functionally equivalent** (same output for all inputs) then the attributionshould always be identical.

Apparently DeepLift and LPR are breaking this axiom nbceause they use mofied form of backpropagation where so computing $\frac{dF(x)}{dx}$ is not equivalent anymore to applying the technique (based on backpropagation / chain rule) so even if they have the same output 2 different models will have different activations.

> All path methods satisfy Implementation In-variance. This follows from the fact that they are definedusing the underlying gradients, which do not depend on theimplementation.

Integrated Gradients (being a Path method) satisfy this axiom because the way we compute backpropagation is not changes so it is equivalent to $\frac{dF(x)}{dx}$.

- Completeness : The sum of the individual differences in contribution should be equal to the total difference of output.

$$\Sigma_{i=1}^{n} \text{IntegratedGrads}_i(x) = F(x) - F(x')$$

This is a desirable axiom in DeepLift and LPR that is supposed to be satisfied in Integrated Gradients

Method

$F : \mathbb{R}^n \rightarrow [0, 1]$

$x : \text{Image} \in \mathbb{R}^n, x' : \text{Baseline} \in \mathbb{R}^n$

If we trace a straight line between x' and x in the $\mathbb{R}^n$ space : $x' + \alpha * (x - x')$ the integrated gradient is the integral of the gradients of output $F(x)$ wrt th input x along this line :

$$\text{IntegratedGrads}_i(x) ::= (x_i - x'_i) \times \int_{\alpha=0}^{1} \frac{\partial F(x' + \alpha \times (x - x'))}{\partial x_i} \, d\alpha \tag{1}$$

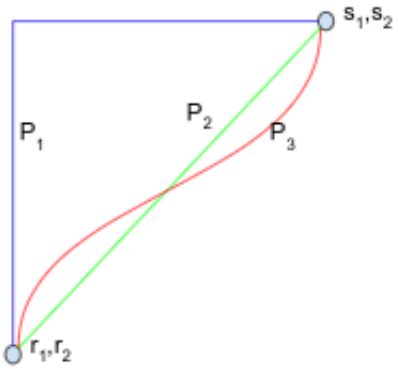


Figure 1. Three paths between an a baseline (r_1, r_2) and an input (s_1, s_2). Each path corresponds to a different attribution method. The path P_2 corresponds to the path used by integrated gradients.

This integral can be approximated using Reiman Sum (Notebook) :

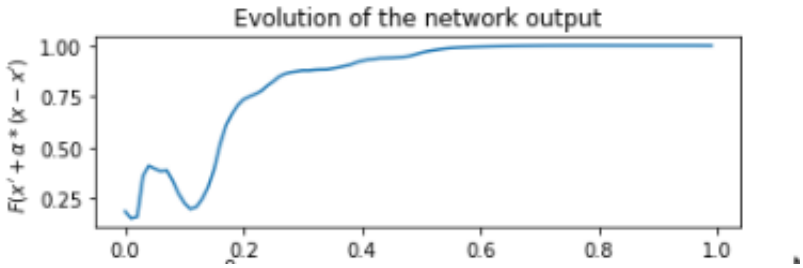
$IntegratedGrads_i^{approx}(x) \triangleq (x_i - x'_i) \sum_{k=1}^{m} \frac{dF(x' + \frac{k}{m}(x - x'))}{dx_i} * \frac{1}{m}$

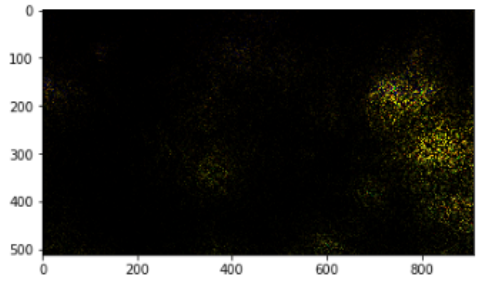
choosing the $m \in [20, 300]$

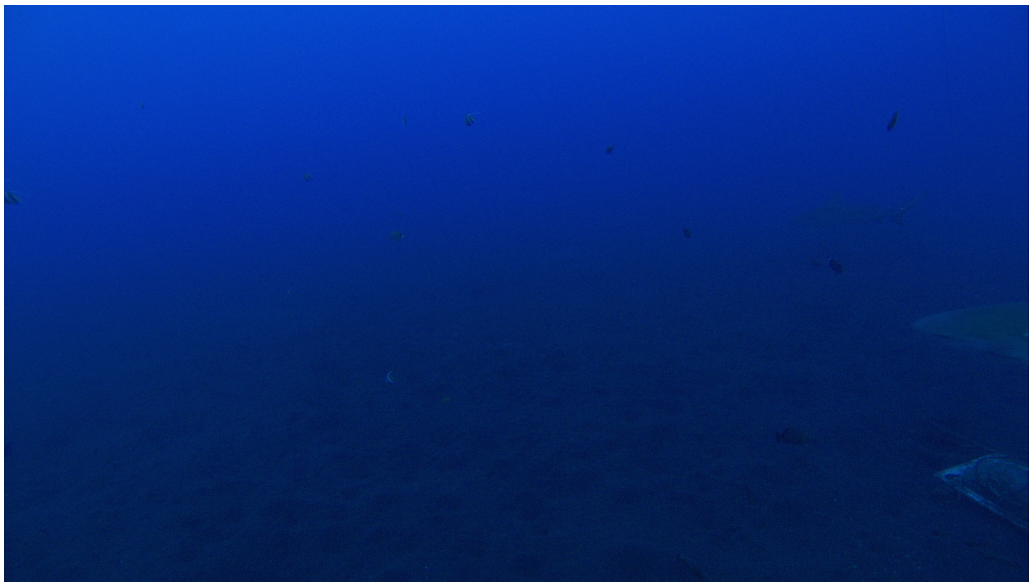
Example Implementation

Manual

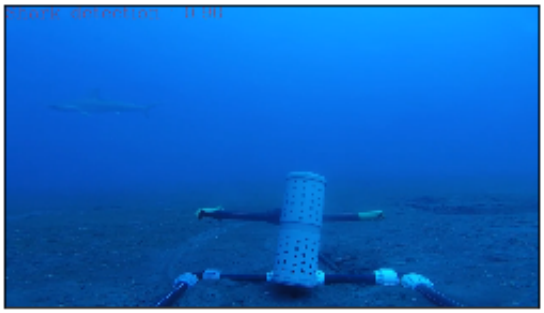
```
baseline =  torch.zeros_like(it)
step_size = 0.01
alphas = np.arange(0,1,step_size)
grad_map = torch.zeros_like(it)

for al in alphas :
    im = baseline+al*(it-baseline)
    im.requires_grad_(True)
    o = sh.net(im)
    o.backward()
    grad_map+= im.grad
```

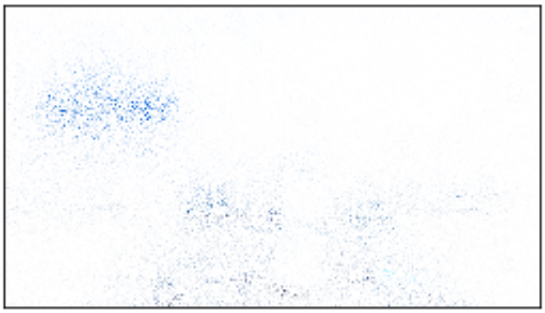


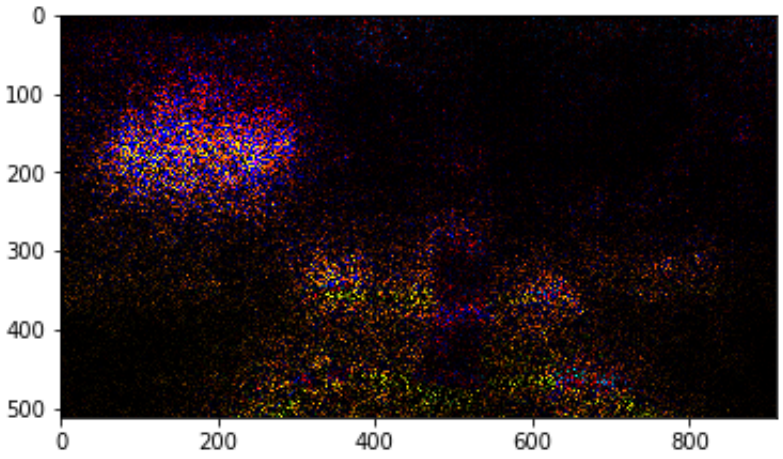


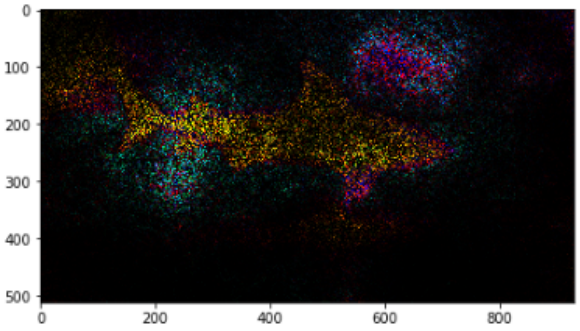


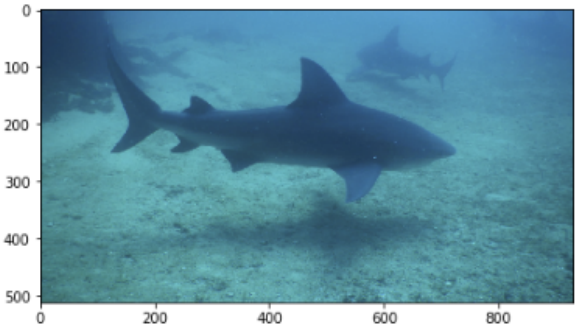
As we can see we are actually mainly variating the "intensity" of each pixel of the image. The resulting map correstpond where the shark is.











You can confirm the convergence by checking the completeness (comparing the sum of the contributions in the grad map and de difference of activations between the image and the baseline) :

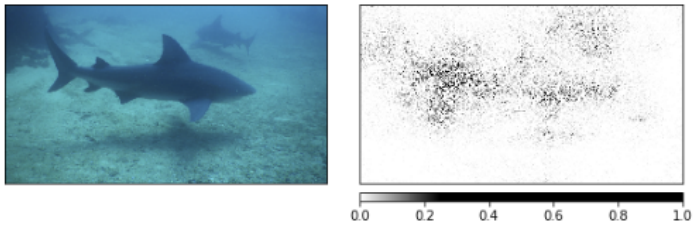
```
grad_map = (it-baseline) * (grad_map*step_size)
d = abs(sh.net(it) - sh.net(baseline) - grad_map.sum())
print('Delta : {:.2f} - {:.2f} = {:.2f}'.format(grad_map.sum().item(),
(sh.net(it) - sh.net(baseline)).item(),d.item()))
```

```
Delta : 7.58 - 7.54 = 0.04
```

Implementation using Captum

```
integrated_gradients = IntegratedGradients(sh.net)
attributions_ig = integrated_gradients.attribute(it, target=0, internal_batch_size=5, baselines=it * 0)
default_cmap = LinearSegmentedColormap.from_list('custom blue',
                                                 [(0, '#ffffff'),
                                                  (0.25, '#000000'),
                                                  (1, '#000000')], N=256)
#attributions_ig = integrated_gradients.attribute(it, target=0)
_ = viz.visualize_image_attr_multiple(np.transpose(attributions_ig.squeeze().cpu().detach().numpy(), (1,2,0)),
                                      np.transpose(transformed_img.squeeze().cpu().detach().numpy(), (1,2,0)),
                                      ["original_image", "heat_map"],
                                      ["all", "positive"],
                                      cmap=default_cmap,
                                      show_colorbar=True)
```

executed in 370ms, finished 20:44:07 2020-04-11



Close methods and choice of the baseline

PDF

Model Agnostic Interpretations Methods

Model agnostic interpretation methods are considering the model as a black box and thus not using any gradients information or other.

Global Surrogate Method

Train a **naturally interpretable and simple model** (Logistic Regression, Decistion Tree ...) to do the same prediction as the model you want to interprete and then interprete this simple model with the calssic ways.

Perform the following steps to obtain a surrogate model:

1. Select a dataset X. This can be the same dataset that was used for training the black box model or a new dataset from the same distribution. You could even select a subset of the data or a grid of points, depending on your application.
2. For the selected dataset X, get the predictions of the black box model.
3. Select an interpretable model type (linear model, decision tree, ...).
4. Train the interpretable model on the dataset X and its predictions.
5. Congratulations! You now have a surrogate model.
6. Measure how well the surrogate model replicates the predictions of the black box model.
7. Interpret the surrogate model.

If the surrogate model is working quite well you can then use it for interpretation, otherwise you will have to find a more complicate model.

Advantages :

- It is intuitive and easy to use
- You can use any surrogate model you want

Disadvantages :

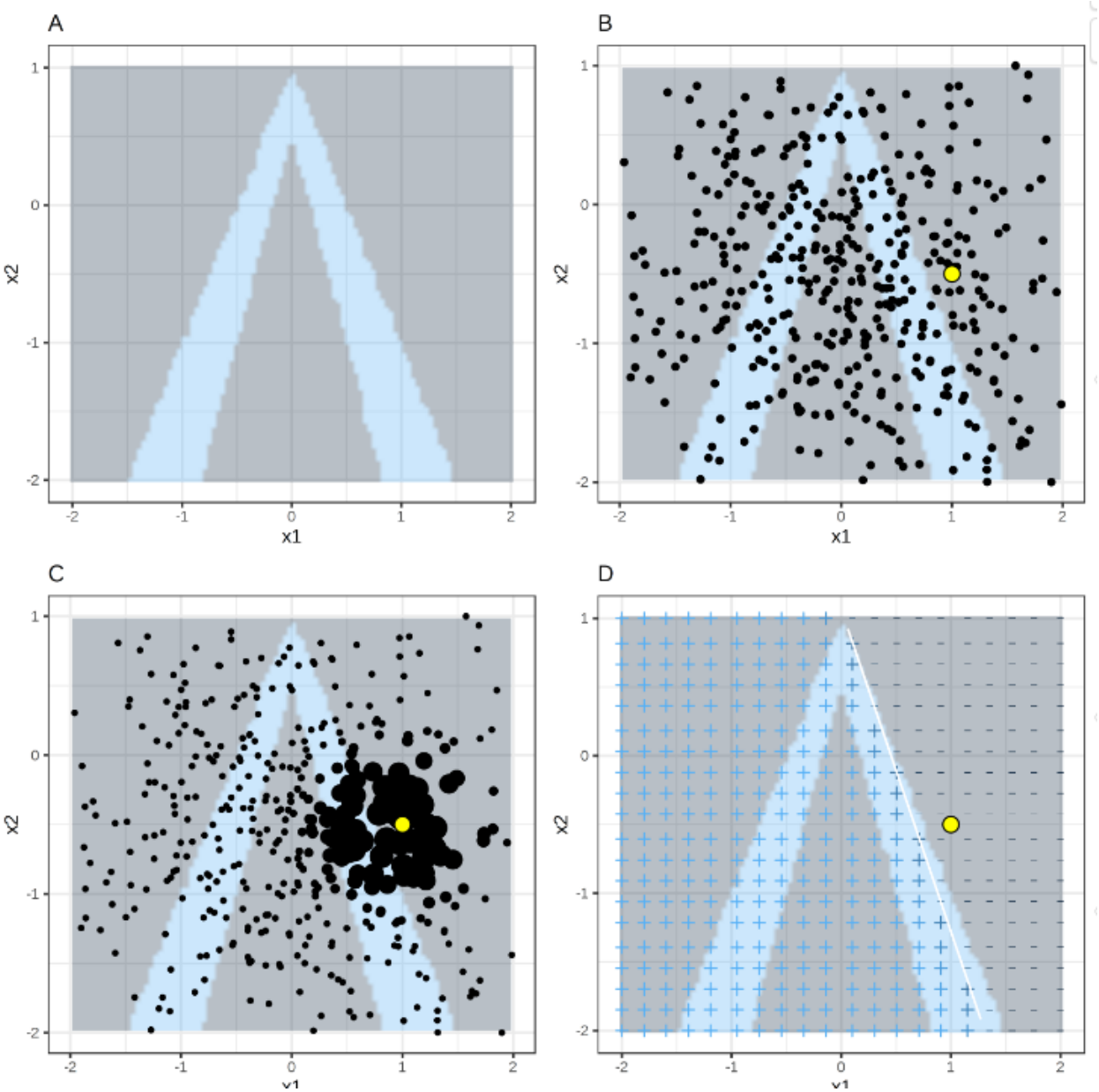
- It is quite hard to set a threshold on a good / less good model to use as surrogate
- Maybe the model will work really well on one subset of data and not another

Local Surrogate Model (LIME) :

This is an alternative to global surrogate model in the case where it is not possible to train a good global simple surogate. The goal here is to train a model that is good at predicting the same thing as the model we want ot interpret around one given data sample, then we can interpret the prediction for this datasample.

The recipe for training local surrogate models:

- Select your instance of interest for which you want to have an explanation of its black box prediction.
- Perturb your dataset and get the black box predictions for these new points.
- Weight the new samples according to their proximity to the instance of interest.
- Train a weighted, interpretable model on the dataset with the variations.
- Explain the prediction by interpreting the local model.



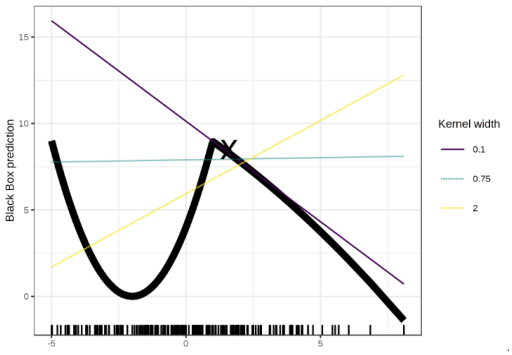
For example, on this fire the black box model prediction are in the background (Blue area for positive and grey for negative). We want to explain those prediction for the point yellow, so we sample the black points from a sample distribution, weight them based on their distance with the yellow point and then train a simple linear model and then plot it's prediction (+/-) signs on the background. As we can see the model is pretty bad globally but quite good locally while being very simple. We can then interpret the model to understand that around the yellow point x_1 have a negative impact on the output and x_2 have a positive impact. It gives a local explanation.

Issue with the distance metric

One of the main issue with LIME technique is that you need a way to evaluate the distance between the black points and the yellow point. And as we know it is really hard to do, especially in lower dimention.

LIME currently uses an exponential smoothing kernel to define the neighborhood. A smoothing kernel is a function that takes two data instances and returns a proximity measure. A small kernel width means that an instance must be very close to influence the local model, a larger kernel width means that instances that are farther away also influence the model.

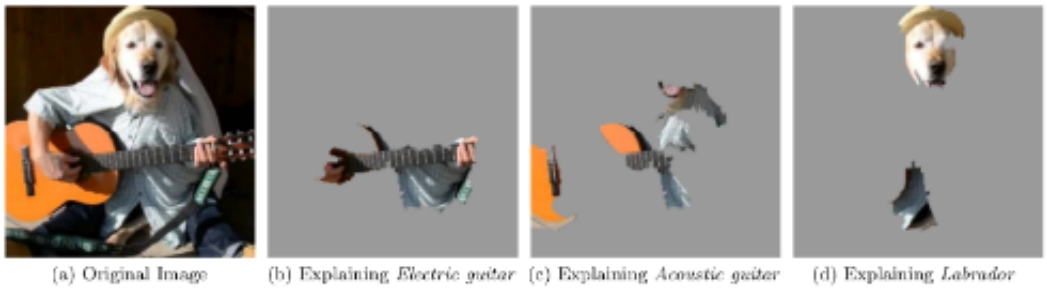
In this setting, determining the size of the kernel is a really hard task and can influence a lot the results.



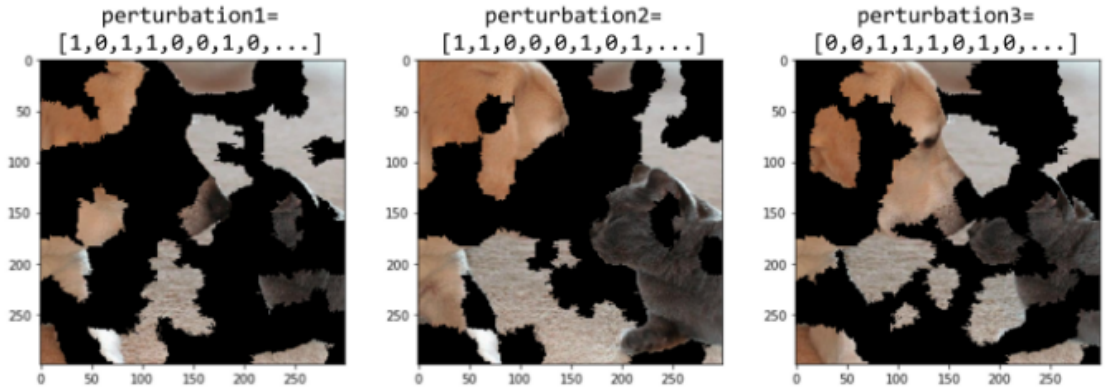
As we can see here if the kernel is very small it will only check the very close points for the classifier (be decreasing in this case) if it is very big it will be more largely aware and then increase,

LIME for images

For image the task of generating close / valid images is super hard to and evaluate the distance is also an open problem. The way lime is working for images is really different from what I understood :

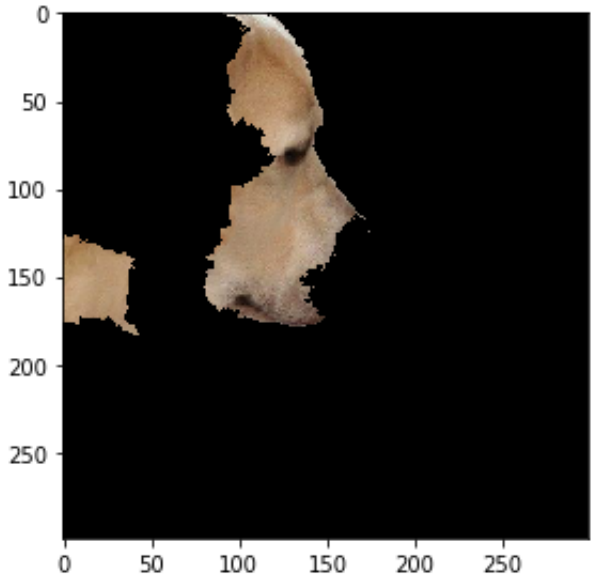

(a) Original Image (b) Explaining *Electric guitar* (c) Explaining *Acoustic guitar* (d) Explaining *Labrador*

1. Generate perturbation for Input images using Superpixel (Group of pixels with the same color) division



2. Predict a class for each perturbation
3. Evaluate the distance of each perturbation to the original input
    - Using cosine distance between the input and the perturbation
    - Apply a kernel on the distances : $\sqrt{e^{-\frac{distances**2}{0.25**2}}}$ to get the weights

4. Use the weighted sample to train a Logistic regression that takes the as input the vector indicating the presence of each super pixel $x = [0, 1, 1, 1, 1, 0, 0, 1 \ldots, 1] \in \mathbb{R}^{super\ pixel}$
5. Extract the weights from the logistic regression and to get the importance of each super pixel and show an image with the top superpixels



Disadvantage of the technique :

- We don't know how the network react to unrealistic images

Sources

[1] Axiomatic Attribution for Deep Networks - Mukund Sundararajan, Ankur Taly, Qiqi Yan

[2] https://christophm.github.io/interpretable-ml-book

[3] Distill Feature Attribution - https://distill.pub/2020/attribution-baselines/