Advanced representation of saliency maps

Context

During the previous lessons, you have manipulated saliency maps from gaze fixation points. You represented them by generating a `heatmap`. In the next sessions, you will use explanation methods that produce saliency maps, and will need to represent them differently. In this session you will generate other kinds of representations that will be presented later.

Constraints

To achieve this goal, you will have to use the Pillow library https://python-pillow.org/ and write a clean libary that you will use during the next lessons. Browse its documentation to understand how to use it.

All functions will be implemented in `representation.py` file. All functions will be tested following the strategy of your choice

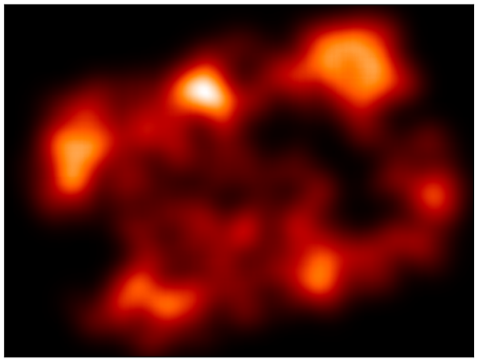
We assume a saliency map is represented by a `Saliency` object that corresponds to a matrix of floats between 0 and 1 for unsigned salliencies and -1 and 1 for signed salliencies (that you may met in following lessons).

Your functions generate `RGBImage` that corresponds to an RGB image generated by the `Pillow` library.

`heatmap` representation

You already have the code from the previous lessons.

Implement `represent_heatmap(saliency: Saliency, cmap: Union[None, Cmap]) -> RGBImage` that produces the image that represents `saliency` as a heatmap. A colormap can be provided optionally to let the user choose the visual. If not, generate a grayscale version for unsigned maps. Choose for a signed one.



$$Heat(x,y) = Colormap\left(Saliency\left(x,y\right)\right) \quad (1)$$

Implement `represent_heatmap_overlaid(saliency:`

`Saliency, image: RGBImage, cmap: Union[None, Cmap]) -> RGBImage` that produce the image that represents `saliency` overlaid on the corresponding test image.

$$Heat2(x,y) = Combine\left(Heat(x,y), Img(x,y)\right) \quad (2)$$

The following libraries are usefull to that:

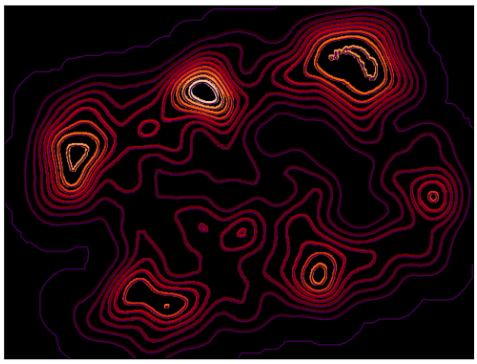
- `numpy` to handle the feature maps
- `Pillow` to handle the generated images and `blend` them
- `matplotlib` to handle the palettes

Implement `isoline` representation

Isolines [1] allow to visualize boundaries of values in the saliancy map by displaying lines of constant values.

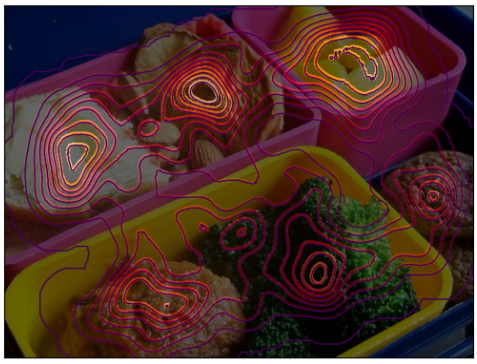
Implement `represent_isolines(saliency: Saliency, cmap: Union[None, Cmap]) -> RGBImage` that produces the image that represents `saliency` using isolines. A colormap can be provided optionally to let the user choose the visual. If not, generate a grayscale version for unsigned maps. Choose for a signed one.

[1] https://en.wikipedia.org/wiki/Contour_line



$$Iso(x,y) = Colormap\left(Countour(Img)\left(x,y\right)\right) \quad (3)$$

Implement `represent_isolines_superimposed(saliency: saliency, image: RGBImage, cmap: Union[None, Cmap]) -> RGBImage` that produces the image that represents `saliency` overlaid on the corresponding test image.



$$Iso2(x,y) = Combine\left(Iso(x,y), Img(x,y)\right) \quad (4)$$

The following libraries are usefull to that:

- `numpy` to handle the feature maps

- `Pillow` to handle the generated images and `blend` them
- `matplotlib` to handle the palettes and compute the isolines

Implement `selection` representation

One can be interested by seeing only the import features (i.e. pixels) and hidding the others.

Implement `represent_hard_selection(saliency: Saliency, image: RGBImage, threshold: float) -> RGBImage` that produces the image that contains only the pixels where the saliency value is higher or equal than the treshold.

$$Hard(x,y) = \begin{cases} Img(x,y), \text{ if } Saliency(x,y) \geq t \\ Black, \text{ otherwhise} \end{cases} \quad (5)$$

Implement `represent_soft_selection(saliency: Saliency, image: RGBImage, threshold: float) -> RGBImage` that produces the image that represents contains only the pixels intensity depends on the saliency map.

$$Soft(x,y) = Img(x,y) * Saliency(x,y) \quad (6)$$

The following libraries are usefull to that:

- `numpy` to handle the feature maps
- `Pillow` to handle the generated images and `compose` or `multiply` them
- `matplotlib` to handle the palettes and compute the isolines

Comparison

Execute all your methods on several pairs of (images, saliency map), using different parameters if any, and visualy compare their results.

- Which conclusions can you draw on their efficiency?
- Is there any winning approach in your opinion?

Extra work

Create additional functions that allow to generate other representations (e.x., side to side images, grids for comparisons, ...).