Deep Learning for Computer Vision
Lab 03

Bolutife Atoki
bolutife-oluwabunmi.atoki@etu.u-bordeaux.fr
Université de Bordeaux

Abstract

This paper contains my solution and results of the third lab session, resources used are listed at the end of the paper

1 Introduction

The aim of this session was to practice how to generate other kinds of representations of gaze fixation points, asides saliency heatmap representation. The representations used include Isoline represenation (Isoline heatmap and Image-Isolines blend), Soft Selection representation, and Hard Selection representation.

2 Methodology

2.1 Isoline Representation

The Isoline representation of a saliency map involves contouring the values in the saliency map using isolines or contour lines., such that these isolines connect points with the same saliency level. The areas with higher saliency values are represented by contour lines that are closer together, creating dense lines, while areas with lower saliency values have contour lines that are farther apart. Also, the colours of the contour lines can be Color / Heat mapped based on the corresponding value of that pixel position in the saliency map. Hence, with this represented density information, viewers can quickly identify which regions stand out the most based on the concentration of contour lines. The resulting Isolines can then further be blended / overlaid on the input image to highlight portions in the image having the most gaze points.

2.2 Selection Representation

Selection Representation is a form of attention visualization that helps represent information in a way that prioritizes certain aspects of a scene or image. It is of two types:

1. **Hard Selection Representation**:
   It involves using binary masks to visualize where to focus attention. It involves a discrete and exclusive selection of a single region or object within the visual field while ignoring the rest. It can be thought of as a spotlight that shines on a specific part of the input, while the rest of the input is masked out. It is obtained by thresholding the saliency map to obtain a binary mask and then superimposing the image and binary mask using the formular:

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

2. **Soft Selection Representation**:
   It involves using a weighted approach to attention via the saliency maps and assigns different levels of attention or importance to multiple regions simultaneously. It is obtained by Superimposing the Image and Saliency map on top of each other, and since the map values range from min to max in order of density of fixations, the same smoothening / widening effect observed in the saliency map would be transferred to the image.

3 Dataset

The dataset used is the Mexculture142 dataset which is made of images of Mexican Cultural heritage recorded during ANR PI Mexculture by IPN CITEDI and gaze fixations data recorded with an eye-tracker at LABRI UMR 5800 CNRS/University of Bordeaux/IPN. The dataset recorded gaze fixations of subjects executing a visual task of recognition of architectural styles of Mexican Cultural heritage. Each category represents different views of the same architectural structure.

The dataset contains 284 samples having 142 subclasses of Prehispanic, Colonial, Modern buildings, we provide 2 examples for each class and the corresponding .txt files of gaze fixations. Also, the saliency map of each image and .txt scanpath files where we have the coordinates and duration of fixations per subject.

The dataset contains 3 folders:
Images: contains 142 categories of Prehispanic, Colonial and Modern styles.
Fixations: holds .txt files which are the corresponding fixations of source images.
Density Maps: contains the subjective saliency maps of each category.

The identifier for each filename is composed as follows:
Images: SSS_XXX_YYY_N_#.png
Fixations: SSS_XXX_YYY_GazeFix_N_#.txt
Density Maps: SSS_XXX_YYY _GFDM_N_#.png

4 Implementation

The code implementation is made up of multiple script files having a main script file (**main.py**) to be called. The other files include:

1. **objects.py**: which holds both objects used:
   - **Saliency:**
     An object of type Saliency, which has an attribute image that stores the saliency map passed in its constructor.
   - **RGBImage:**
     An object of type RGBImage, which has an attribute image that stores the rgb image passed in its constructor.
2. **representations.py**: which holds the various representation functions used in the project.
   - **represent_heatmap()**:
     This takes in as parameters the saliency map (of type Saliency) and the desired colormap to be used (of type str), and returns the input saliency map represented as a heatmapped image according to the specified color map, such that the output image is of type RGBImage.
   - **represent_heatmap_overlaid()**:
     This takes in as parameters the saliency map (of type Saliency), the image (of type RGBImage) and the desired colormap to be used (of type str), and then applies the specified colourmap to the saliency map to get a heatmap (by calling the **represent_heatmap()**) function. The heatmapped saliency map is then blended with the input image by an alpha parameter that controls the blending, and finally returns this blended image as type RGBImage.
   - **represent_isolines()**:
     This receives the following arguments; the saliency map (of type Saliency) and the desired colormap to be used (of type str). The function initially normalizes the saliency map and then uses the **find_contours** of skimage's measure library to implement the marching squares algorithm which finds pixels with constant valued contours in the saliency map at different search levels / ranges. The coordinates for the contours at the different levels are then looped through and an empty image array same size as saliency map has its pixel values at those coordinates set to the corresponding values of the saliency map to make the isolines. Finally, the obtained isolines image is then heatmapped with the specified colour map (using the **represent_heatmap()** function) and is returned as an image of type RGBImage.
   - **represent_isolines_superimposed()**:
     This takes in the saliency map (of type Saliency), the image (of type RGBImage) and the desired colormap to be used (of type str), and initially calls the **represent_isolines()** function to obtain the heatmapped isolines image, then blends this with the input image, such that the blend is also controlled by parameter alpha. This blended / overlaid image (of type RGBImage) is then returned.

- **represent_hard_selection()**:
  This takes in the saliency map (of type Saliency), the image (of type RGBImage), and a threshold value and then obtains a binary mask by thresholding the saliency map values based on the specified input threshold value. This mask is then overlaid on the input image to highlight only the portions with saliency values higher than the threshold, with other section values set to 0. The resulting image is of type RGBImage.
- **represent_soft_selection(saliency, image)()**:
  This takes in the saliency map (of type Saliency), the image (of type RGBImage), and then returns the input image masked by the saliency map, such that only the pixels with gaze points density are shown, with the others set to 0. The resulting image is of type RGBImage.

3. **utils.py:**
   which holds the helper function used in the project.
   - **normalise()**:
     It takes in a matrix and returns its normalized version (to range 0 - 1) by dividing each element in it by its max value.
   - **grid_layout()**:
     It takes a list of images and a list of string titles as inputs, and plots & shows these images in a grid and saves the grid image.
4. **main.py**
   It is the main python script for this project and it can be called directly from the command line, it has a main function which accepts the following arguments when called using argument parser;
   - path to test image
   - path to saliency map
   - the display type; **grid** which shows a grid of all results or **singles** which shows the results individually.

   An example format of calling the function is shown below

```
    python main.py --test_image_path='test.jpg' ...
        --test_saliency_map_path='test_saliency_img.png' --display_type='grid'
```

   The main function does all the necessary imports of the utils, representations and objects files and then calls the functions for the implemented representations, and finally displays the obtained images either as a grid or individually, depending on the type specified with the grid display set as default.

5 Results

The results of the functions implemented are evaluated visually by plotting all obtained images in a grid and analyzing visually. To this end, the given test image (and its saliency map) as well as 4 randomly selected images (and their saliency maps) were tested and plotted, shown below:

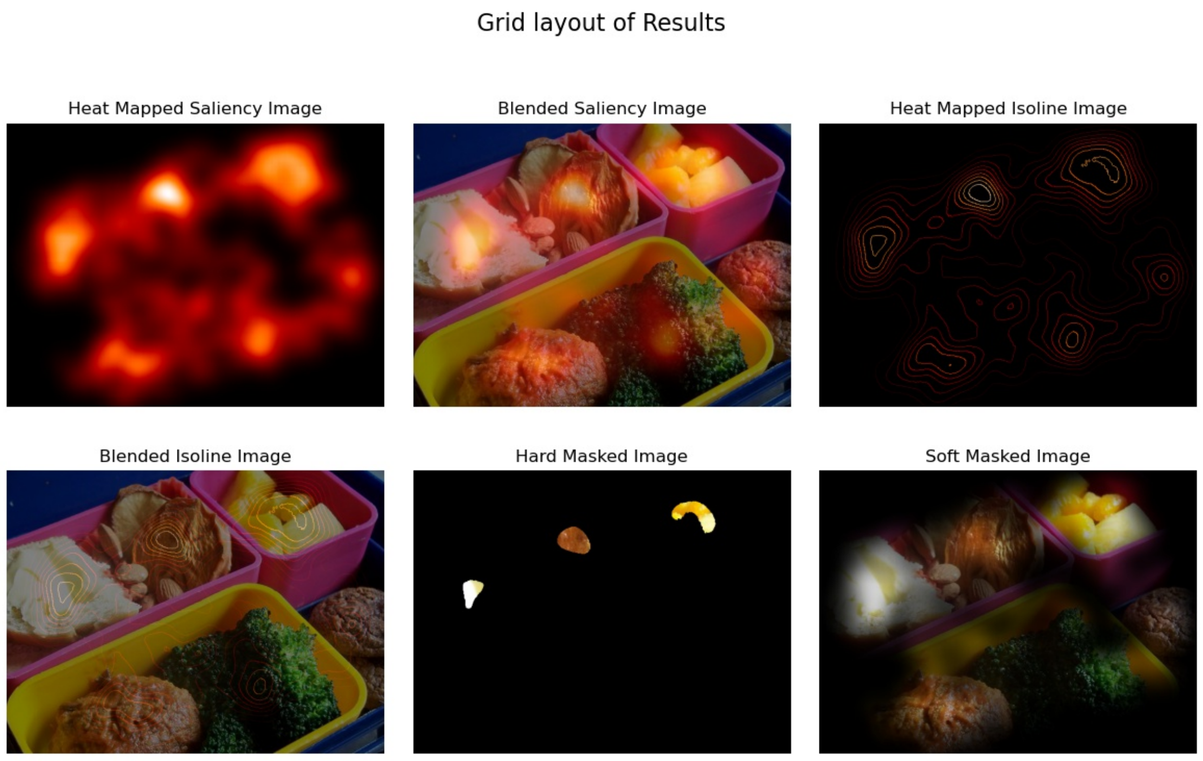

Figure 1: Grid layout showing heatmapped saliency and isoline images, blended saliency and isoline images, hard / thresholding representation and soft representation of fixation points

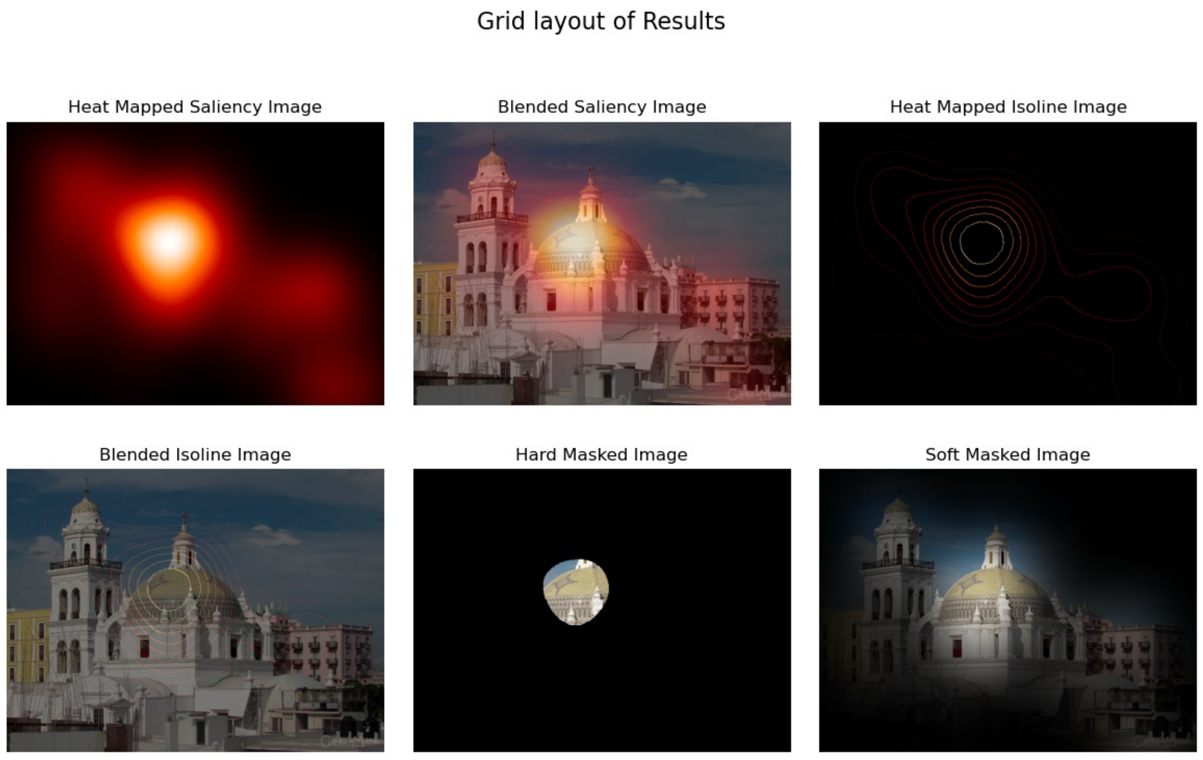

Figure 2: Grid layout showing heatmapped saliency and isoline images, blended saliency and isoline images, hard / thresholding representation and soft representation of fixation points

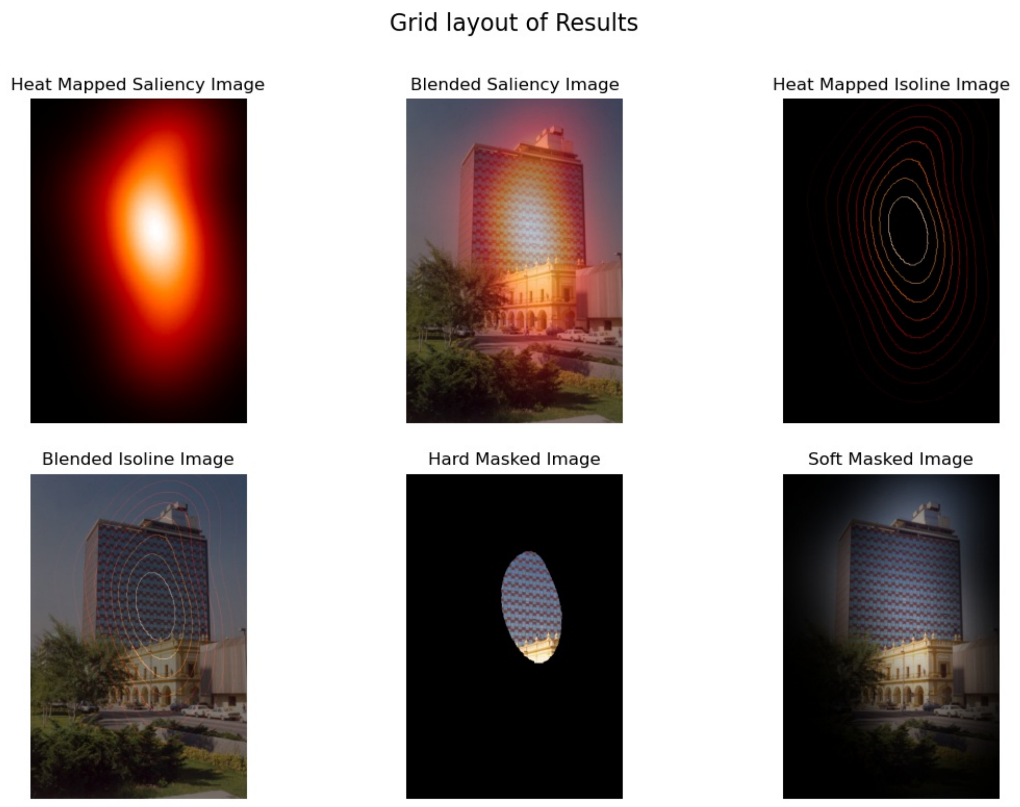

Figure 3: Grid layout showing heatmapped saliency and isoline images, blended saliency and isoline images, hard / thresholding representation and soft representation of fixation points

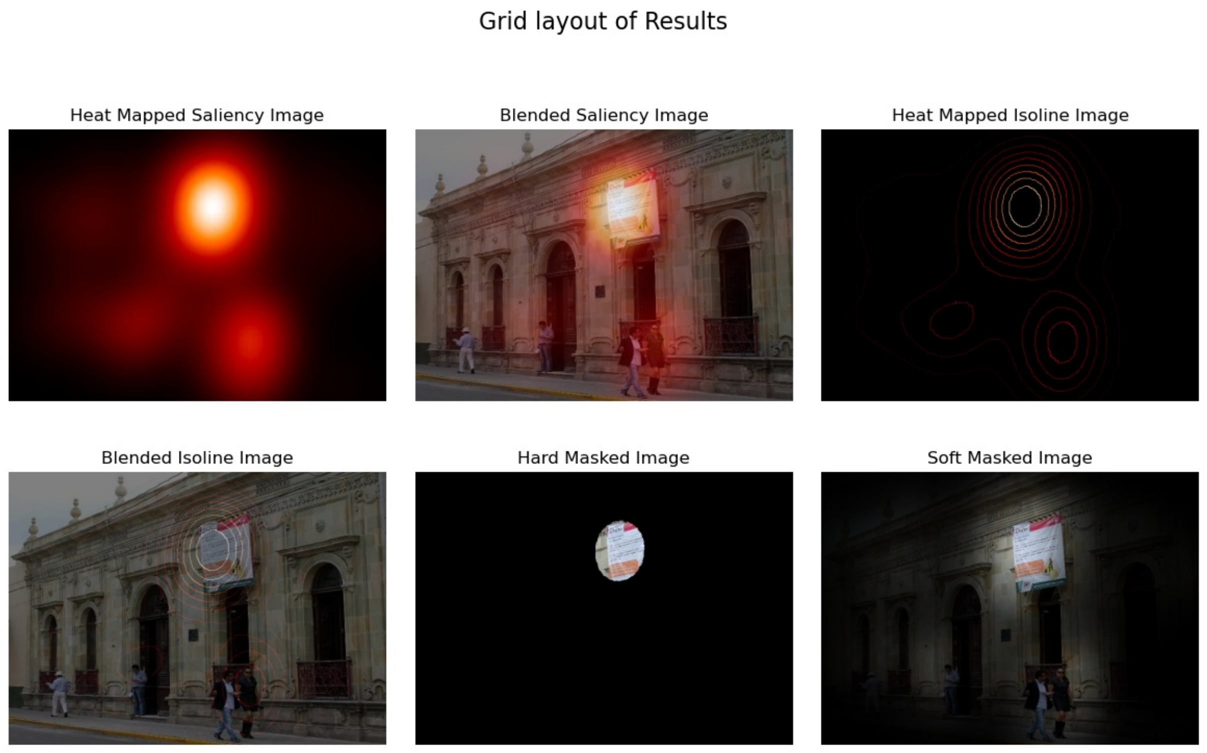

Figure 4: Grid layout showing heatmapped saliency and isoline images, blended saliency and isoline images, hard / thresholding representation and soft representation of fixation points

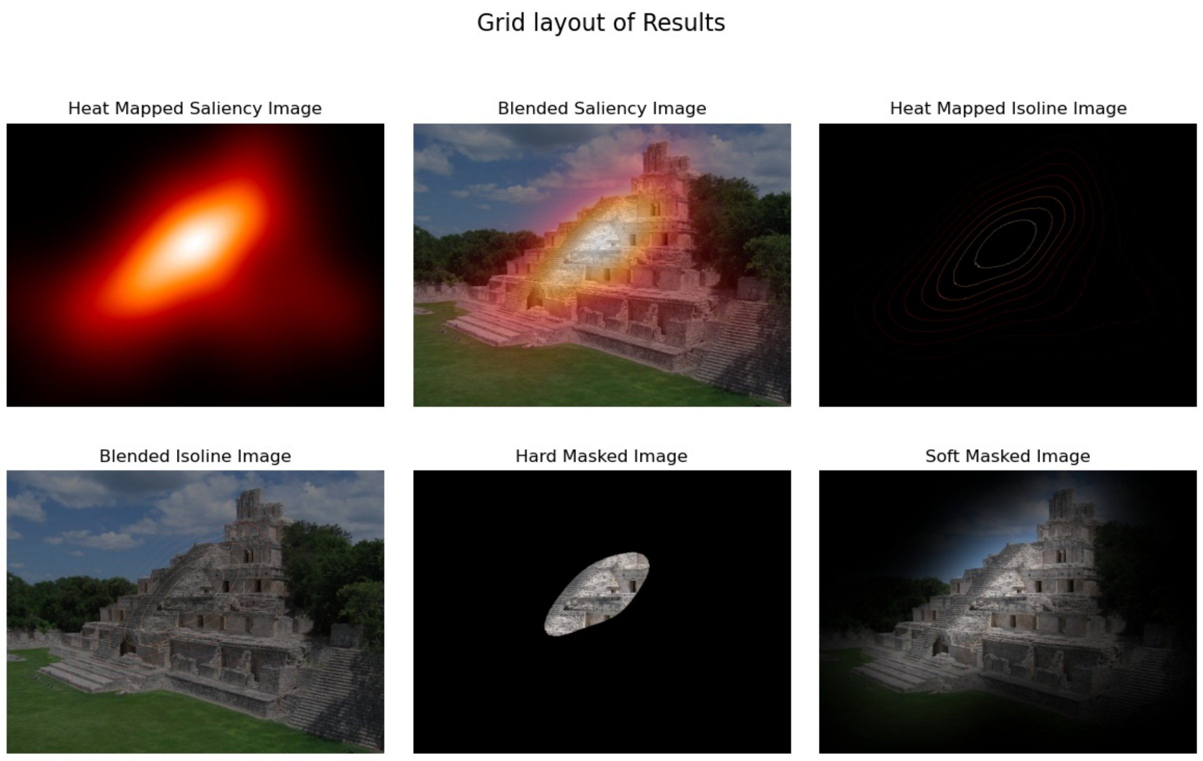

Figure 5: Grid layout showing heatmapped saliency and isoline images, blended saliency and isoline images, hard / thresholding representation and soft representation of fixation points

6 Discussion

Upon analyzing the grid images obtained, it was observed that the implentations of the functions worked as specified in the task description and 6 forms of gaze point density representations were obtained. It should be noted that optimizations were implemented and skimage's measure and find_contours was used instead of matplotlib's as it provided more control over the generation of the contours as the values could be obtained and then manipulated instead of the image plot which is obtained using matplotlib. It also allowed to see and understand the marching squares algorithm which matplotlib did not afford.

7 Conclusion

The following were implemented with visually correct results; saliency heatmap representation, Image-Saliency map blend, Isoline heatmap representation, Image-Isolines blend, Soft Selection representation, and Hard Selection representation.