Image finder – User study
Software specifications

Document purpose

This document specifies the goals and aims of a user study software. It also touches technical solutions required to implement the software.

Software details

This software presents a pre-sorted matrix of images to a test user. The user is shown a preselected image for a few seconds and is then tasked to find this specific image during a limited time period in the matrix of the images displayed on the screen.

The application then collects data about the user behavior during the test session. The data collected will be but are not limited to:

1. The image was correctly found or correctly marked as “not present”.
2. The time in which the user provided an output (clicked an image, marked as not found, etc.).
3. Dimensions and other details of the data displayed (size of images, number of images shown, number of columns, etc.).
4. Which actions the user performed (scrolling, hovering the cursor above some images, etc.).
5. In which format were the data presented (read further for more details).

Possible implementation of this data collection is mapping the actions of the user to some event IDs and collecting them in a simple log.

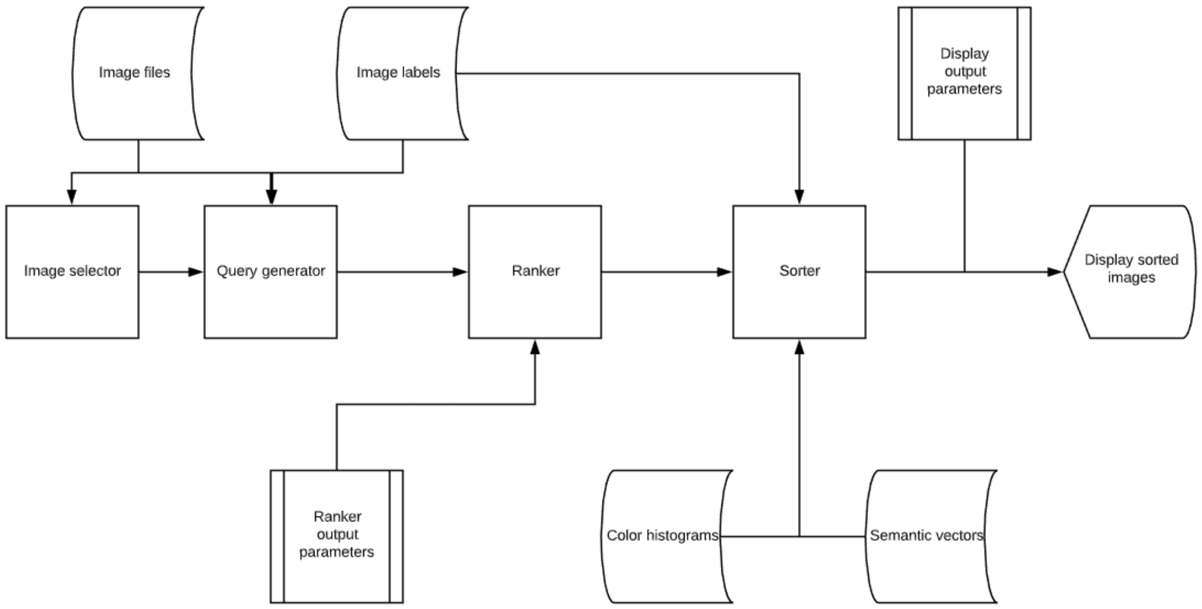
Data presentation

The data will be prepared in the backend and the user will then receive a matrix of images in a browser. These images will be sorted by some sorting metric. Examples of these sorting methods are but are not limited to:

1. Images are sorted according to relevancy from left to right, starting at the left upper corner (most relevant).
2. Images are sorted diagonally according to relevancy from left upper corner (most relevant) down to right lower corner (least relevant).
3. Images are clustered according to color histograms. The sorting is done by a self-organizing map algorithm.
4. Images are clustered according to their semantic vectors. This sorting is done by a convolutional neural network which accepts semantic vectors.

Data pipeline

Before this document goes into more details about the types of sorting and how to achieve them, the pipeline of the data for the matrix of images needs to be described.



Every session begins with selecting a random image from the image files. The program then looks up the labels for this image and from these labels, creates a semi-random query (more on that later) in the query generator. This query is then sent to the ranker sub-program which selects top N candidates for the query (N is provided to the ranker in parameters). This is then sorted in the sorter sub-program which then sends to sorted data to the front-end.

Query Generator

In this project, the users will not be able to select the query parameters by themselves. They will be pre-selected for them. This study will primarily focus on how easily the users are able to find a pre-selected picture in a given matrix of images.

Allowing the user to make a query on their own is a possible upgrade of this application but is not the target now. Possible implementation of the user query input is showing an image to the user and then allowing them to create the query on their own. They will then get a matrix of images created based on their query. This case, however, focuses more on the queries instead of the image sorting.

How the query generator is going to generate the queries from the labels is a thing to be experimented with since choosing top three labels might be way too easy for the user to find the image. How this problem has been handled will be described in the technical documentation.

Relevancy sorting

As mentioned above, one of the possible sorting options is sorting the images by relevancy (options 1. and 2.). This will be done by using labels for each image.

The labels (tags) for every image are already obtained and therefore, there is no need to generate them. Every image is described by a vector of labels whose values are real numbers from an interval [0, 1] where 0 means zero relevancy and 1 means absolute relevancy.

Example: We have 5 possible labels ([orange, beer, beach, snow, sun]). The label vector of an image is [0.7, 0, 0.7, 0.1, 1] which means sun is very relevant, beer is not relevant at all, etc.

The sorter will then sort these images according to the sum of queried labels. The highest sum will be in the left upper corner and the rest will vary according to whether method 1. or method 2. Is used.

Color histogram sorting

Another possible sorting is getting the output from the ranker and sorting it with self-organizing map algorithm using the color histograms. This will cluster similarly colored images together which might help the user to find the preselected image.

The histograms will be generated by Numpy or OpenCV Python libraries.

More on color histograms here.

Semantic vectors sorting

Like color histogram sorting, the self-organizing map will cluster the images according to their semantic vectors.

These semantic vectors will be generated by convoluted neural network using tensorFlow Python library.

More on semantic vectors generation here.