Matthew “Cookie” Davis

CSC 664

Multifaceted Hyperlinked Images

When you look at a picture, you don’t simply see a single image in an isolated state. You see all the different parts of it and how they are related to things you’ve seen before. You see references to other images you have, and you mentally make the links between one image and the next. The ability to map these links has many practical and theoretical applications. For my project, I’ve taken the ability to map these regions on images and make these connections, then recall and traverse these made links to see the web of connections and how images are related. Throughout my proposals, I’ve shown how this is usable for artists and tailors, connecting together references for different styles and ensembles so that a single person can quickly pull together important details for projects simply by selecting a region on one image.

## The Problem

To solve this problem takes several different components:

* Upload images.
* Define facets (labels) that are defined by the user.
* Draw regions on the image.
* Apply a facet to each region on an image and keep them persistent.
* Connect images through user defined links.
* Draw these user defined links based off of a selected region.

By defining the steps of the problem in this fashion, it becomes easier to visualize each of the moving parts needed to go from seeing an image to connecting it to other sections of other images. Additionally to make this all easier, the input and output of this problem is handled through a web based UI.

## The Sky View

To go about solving this issue of linking multiple regions of images, a lot of information is input by the user with information stored and manipulated through a backend API. Each major component (images, regions, and facets) are stored as individual objects within a database.

* Images are simple objects that know what they are, what image they hold, and keep track of a set of images that they are linked to (which is updated and added to by the user through the UI).
* Facets are user defined labels, knowing what they are and their label and nothing more.
* Regions are more complex objects. A region maintains its dimensions and holds a foregin key relation (one to many) with an image, and one with a facet. Regions act as a user defined through table, linking images to facets for the purpose of database lookups.

As the database is populated by these major components, the majority of the problem solves itself. When a region is made, the user is prompted with a list of all images that have one or more regions on them that share the facet of the region, and from there are able to populate the current working image’s linked list.

When a region is selected after creation, it then becomes the focus for creating a graph. If a region is used as a search, it becomes the central node for searching the database. Within the backend, the central image’s linked list is filtered down to images that have a region of the selected facet. Each image is then entered, made into a node, and this process is repeated, until a full graph of nodes and links is generated through the backend, and then displayed as a part of the front end.

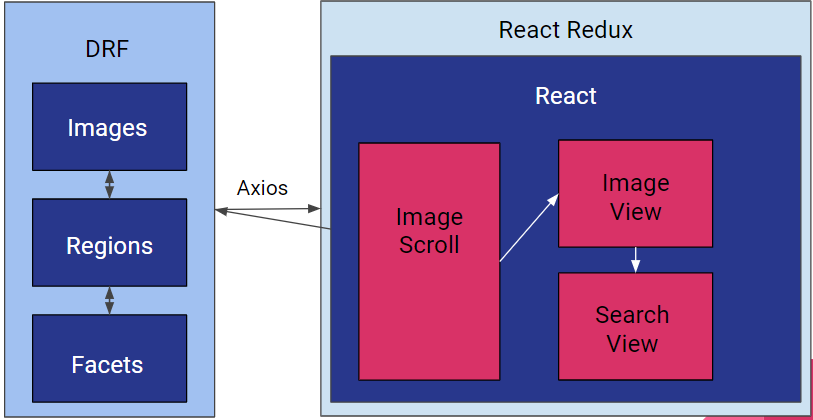
## Prior Work

The work that I studied while working through this problem revolved primarily around two papers offered by the professor: “Multifaceted Hyperimage-Based Organization and Interaction with Bio-Medical Images” and “Characterization and Analysis of Emergent Image Semantics Using Network Models”.

While the latter of these two papers offered several good points of information in regards to the use of this information. The means of seeing how users section and connect images and the emergent semantic models that come from analyzing these links is incredible information. However, it was the former on bio-medical images from which I drew a fair amount of my inspiration. Primarily the use and importance of a UI to interact with the problem itself, and the use of displaying all the information in a comprehensible manner.

However, while much of the information in the paper was stored in header fires for each image and used XML server commands, I separated out the majority of the components into their own data structures within a dedicated backend. This allows for more data extrapolation and allows each object to be modeled and modified individually, though there is a lot more overhead in maintaining a full Django API.

## The Work

The system is as described below, and their interactions are shown in this diagram:  


Breaking down each component and its functions, we have:

* React: The React App is the rendered front end of the application. Through this, components are mounted and the user is capable of interacting with the backend API.
* React Redux: The Redux store is a wrapper around the React App, offering a means of sharing global state values across all the components. The use of the Redux store is to more easily share current working variables between components, a process that can be difficult when needing to pass information up to parents.
* Image Scroll: The Image Scroll is a simple container that displays all images that have been uploaded to the server. From here, the user can select an image to view, which is passed to the Image View described below.
* Image View: The Image view is where a user interacts primarily. From here, the user can interact with the image, primarily through adding/interacting with regions on the image. The Image View displays the current working image and all regions drawn upon it, each as a clickable button. When a region is clicked, a user may initiate a search from it or alter it. Should a user choose to search from a region, the current image and the facet of the selected region are given to the Search View described below.
* Search View: The Search View is primarily a container for the graph. Within it is a component that describes a d3 program to draw the resulting graph from the search. Each image is used as a node, and each link is drawn attached to one another, converted into an interactable SVG image, and displayed within the Search View. This graph is removed and rebuilt whenever a new search is initiated, with the backend also doing clean up to ensure prior graphs do not clutter the system.
* Axios: Axios is the bread and butter of connecting the two frameworks. While it is not a component I built, it is used heavily throughout the react app whenever a call to the DRF backend is required. Primarily, axios also helps to relegate the heavier computational needs to the backend, meaning the frontend app doesn’t need to worry about storing, sorting, or filtering items to reduce its latency.
* DRF: Django Rest Framework, an extra layer of usability atop the Django backend API. This offers a means of storing and retrieving data models in complex ways. The Django backend performs much of the filtering and computation for the app through GET/POST actions tied within its views. Through the framework, images, regions, and facets are used to filter down data sets into relative searches. This is the driving power behind generating the graph, offering images to link, displaying regions, and the whole computational foundation.
* Images: As described, Images are uploaded by the user through the React application. Images are simple data objects, storing an image, and keeping a many2many table of images each are linked to as defined by the user.
* Regions: Regions are drawn sections of an image defined by a user. The region object stores its dimension and location so it can be drawn again, a foreign key relation to the image it was drawn on, and a foreign key relation with the face assigned to it. Regions act as a form of user defined through table, holding together images and facets and allowing the two tables to see each other by looking through the relations of a region.
* Facets: Facets are the simplest yet a vital object. Facets are user defined labels for sections of an image, and are the foundation upon which filtering happens.

## Conclusion

Finally, the results of the application are something of a mixed back. The graphing algorithm is slow but accurate, and there are a few bugs persisting due to asynch errors that require more gentle tuning. However, this is not the end of this program. Some finer tuning on the existing components will grant it greater stability and allow for more ease of use.

Additionally, there are many features that could be added onto this program in the future to allow for it to become more marketable. Adding a user login to allow for multiple users will create a platform that can be hosted. Allowing for further refinement of the search facets, and giving regions multiple facets will help with user traversal of their own data. And finally, implementing a form of machine learning to scan uploaded images and offer regions and links for a user to help automate the process of applying the user’s defined image semantics to images.

## Bibliography

R. Singh, C. Massa and P. -. Corlobe, "Multifaceted Hyperimage-Based Organization and Interaction with Bio-Medical Images," Twentieth IEEE International Symposium on Computer-Based Medical Systems (CBMS'07), Maribor, 2007, pp. 153-158, doi: 10.1109/CBMS.2007.74.

Singh, R, Nakata, R, and Downs, J. "Characterization and Analysis of Emergent Image Semantics Using Network Models." (2011): 411-18. Web.

Venkatagiri, Seshadri, Chan, Mun, and Ooi, Wei. "Automated Link Generation for Sensor-Enriched Smartphone Images." *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 12.1s (2015): 1-25. Web.