Image Labelling App: Demo Design Report

Name: Ruixin Cheng

This document is to introduce the system design of Image Labelling Web App. It will discuss the source of images for being labelled, web pages for showing and labelling images, ways of storing image-related data & labelled results in server, and the work items/milestones that need to be done step by step. However, this document might not be strict as I am still thinking about what good designs could be implemented.

1 Images

There are 3 main concerns regarding to the source images.

1.1 Source of Images

For the time being, the web app will use a certain number of photos, let's say 100, stored in Node.js server locally to serve as raw data waiting to be labelled/commented. Later, third party API, such as Unsplash and Flickr API, could be used to get images that need to be labelled. I currently registered an Unsplash account for getting images but frequency of accessing the API is limited to 50 images per hour. I tried to use Flickr api but account registration failed for no reason. I might try them later.

1.2 Image Category

Second thing that needs to be considered is that random images still need to be categorized. The current images whatever they are from third party api or are stored locally are entirely random types. **- ! edit -** There is no specific category for distinguishing one from another, It should be something like that a bunch of images would bse displayed to users if they are to label football images(e.g. Do they contain football?) and those images could contain irrelevant content. For example, users are verifying images that contain lion but there should be some images that are irrelevant, let's say having African landscapes or other species.

1.3 Image Caption

One thing we need to concern is how we could get caption or description of an image.

1.3 Privacy

Another thing that needs to be concerned is image privacy. Some images might contain privacy issues that need to be mosaicked. I am not really sure how to do mosaiking automatically. There should be some researches on it.

2 Web Pages

There would be several pages for showing web content:

- On the top of window or side of it, there will be navbar/side bar for navigation between pages
- Login & Signup pages will be displayed when users need to login/logout/signup
- A home page will be displayed as index page which leads to image labelling task pages.
- A page for showing users' achievements will be displayed

2.1 Nav bar

A navigation bar on the top of window or side of it is needed mainly for showing links to home page, achievement page and user registration page.

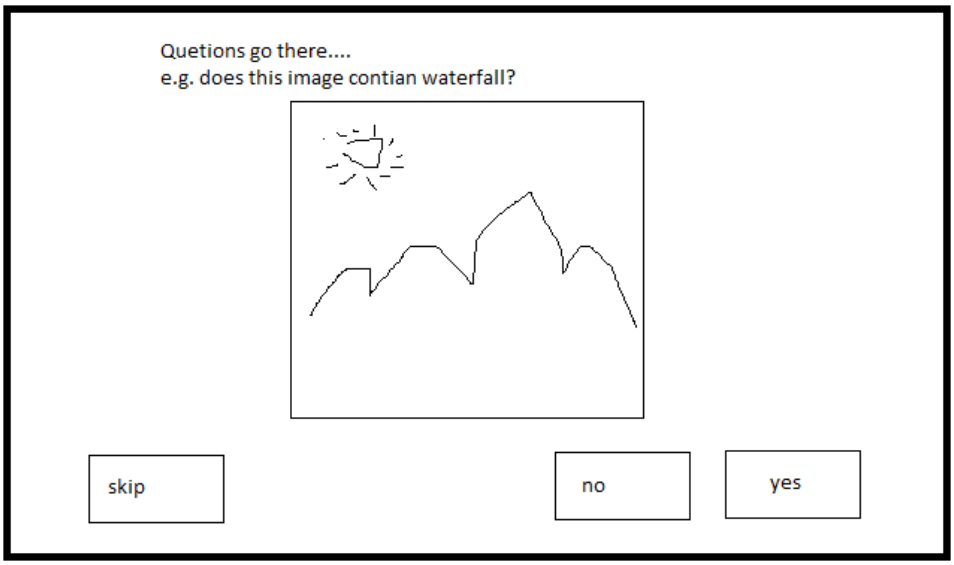
2.2 Registration pages

There could be two pages for showing forms of login and signup separately.

2.3 Home Page

This is main web content of this web app. At '/' page, it will show all image tagging tasks one by one. Going further, there would be '/:task' page for showing specific tasks users can play with. For example, '/imagelabelling' could allow users to judge whether a page contains the object that the task asks. Currently, there will be '/imagelabelling' and 'image-captioning' task pages.

Web layout of image tasks



2.4 Achievement page

This page will show what how many tasks users have finished so far.

3 Data structure of Images

3.1 Choice of Database

Using Key-value database to store training data is recommended. It does not get involved with transactions or other secure operations where MySQL/PostreSQL would do. Data value mostly is relatively at low level and its scale might be expanded potentially. Key-value database fits into place. For the time being, web app could run on a local MongoDB app for testing and demonstration purposes. Later, it could be connected to Azure Cosmos DB. There are some useful links for cloud-based database configurations down below.

Connecting existing MongoDB app to COSMOS DB

Azure Cosmos DB's API for MongoDB

3.2 Data structure / schema of image data

Schemas of Image and User are as follow. Everytime when users post a request(yes or no) to current image, all the server needs to do is to find that image by 'url' and push the boolean value to its related task. I did research on Google Crowdsource and thought that the repeated post request on the same image might not cover the old post request by the same user. It will simply push a new

value to the array instead. Otherwise, it will need to keep track of all users' submits and edit their old submit, which might be a little bit time and space consuming.

Image:

```
{
    url : String,
    tasks : {
        'doesContainXXX' : [Boolean],
        'isGoodCaptionOfXXX' : [Boolean],
        ...
    }
}
```

User:

```
{
    username : String,
    hashedPassword : String,
    Contributions : {
        'imageLabelling' : Number,
        'imageCaptioning' : Number
    }
}
```

4 Work Items

Preparation:

1. Set up react and a node.js server
2. Host 100 images on server for the time being
3. Find a reliable API for getting random but categorized image in long term(this can be done later)
4. Connect to MongoDB locally which will be cloud-based later on

Clident side implementation:

1. Implement a navbar
2. Implement frontend pages and its relevant functions: home page, task pages
3. Implement frontend pages and its relevant functions: registration page, achievement page
4. Put links to web pages in navbar

(During the development, we may need to set up routes/apis for requests between React and Node.js if needed)