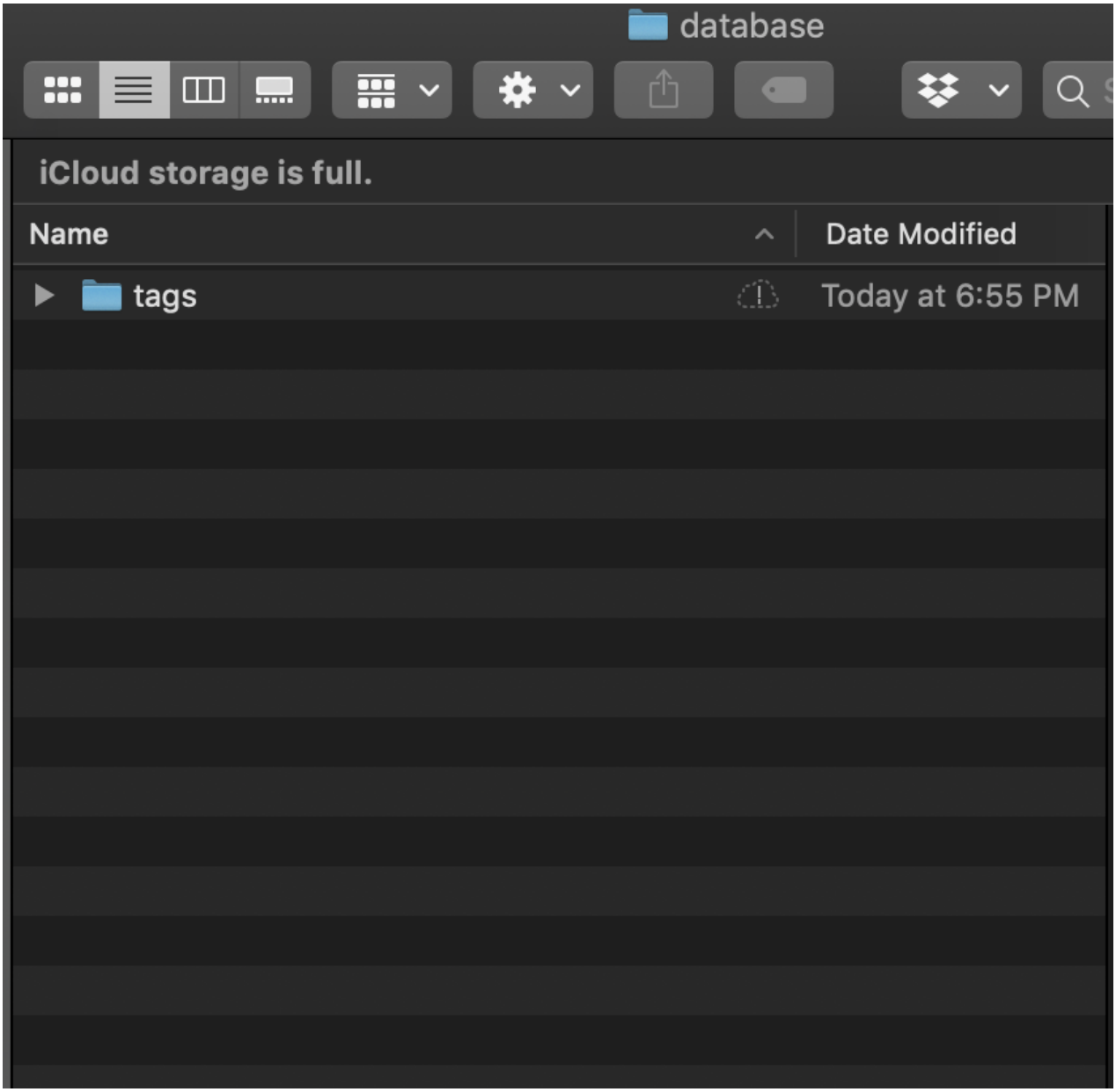
HCI Assignment Two

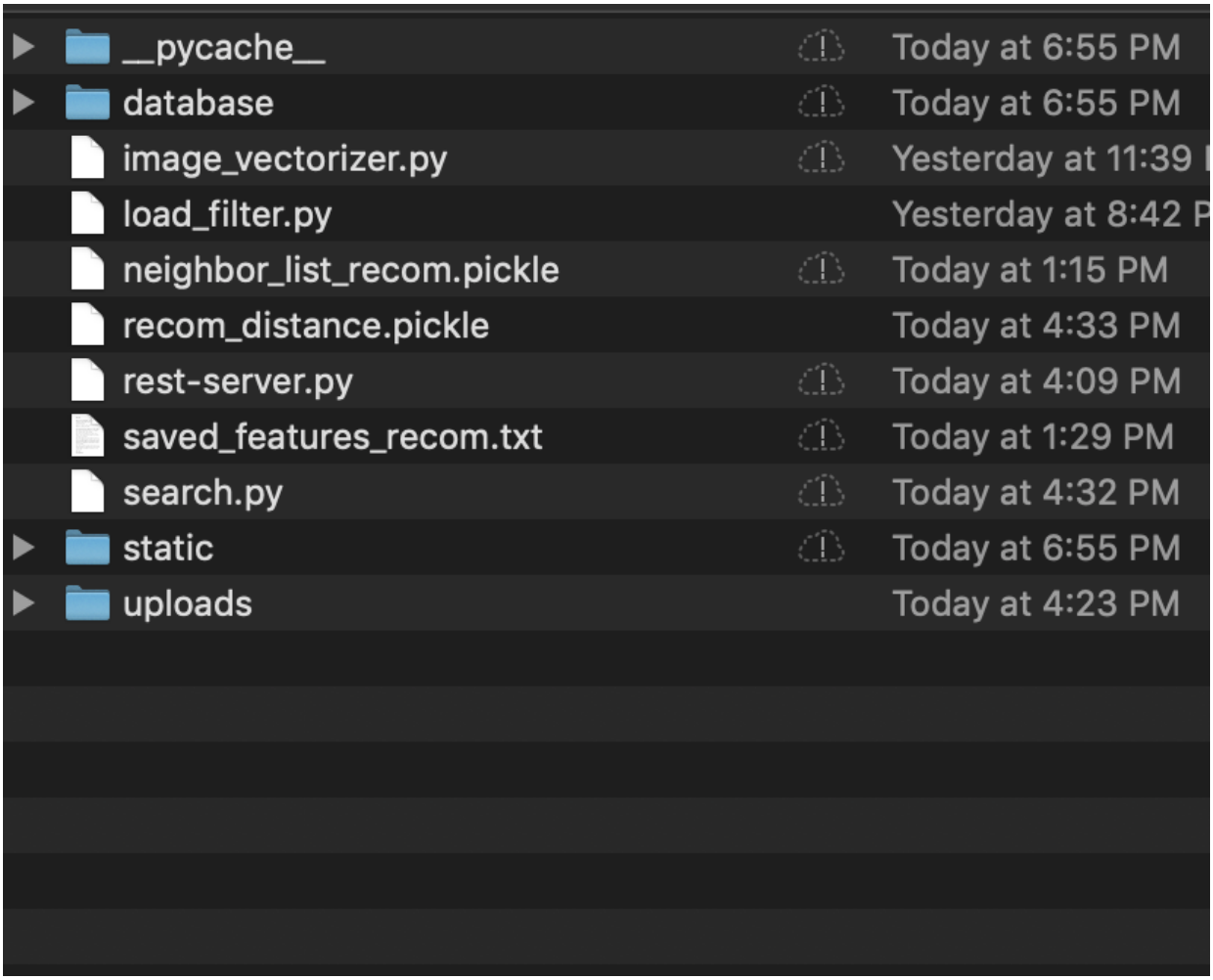
1750114 程子鸣

Overview

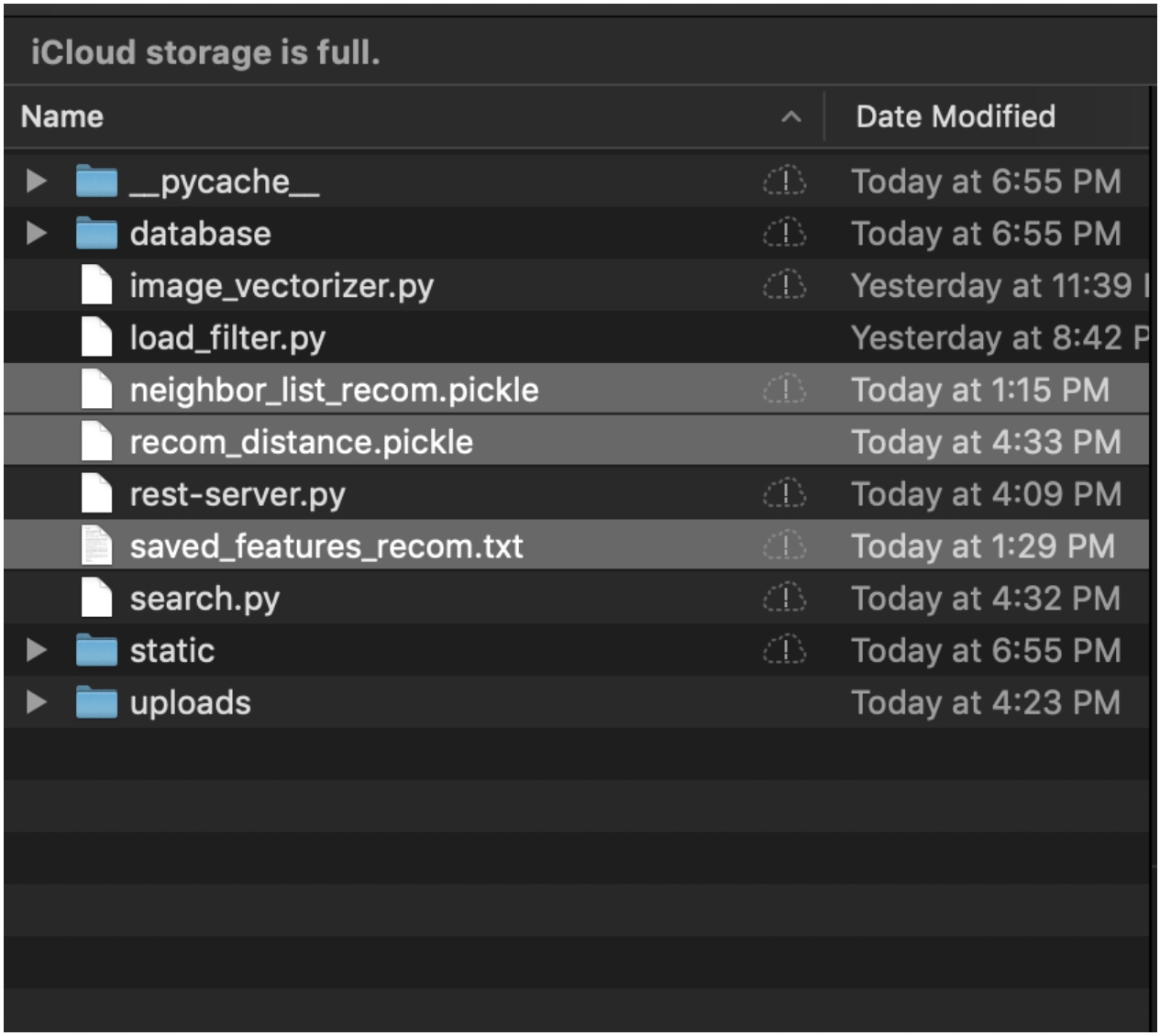
Note!!

Since some of the datasets are very large, I deleted the image database and the imageNet folder in order to keep the file size small enough.


database
iCloud storage is full.
Name
Date Modified
tags
Today at 6:55 PM



I also deleted the pickle file and saved features, Please regenerate them



Basic requirements

Users can use this system to search for similar images with specially designed constraints and then save the images they like.

- Fluent UI design
- Relatively quick feedback
- Reliable results

Five-Stage Framework

Before elaborating all my design of the five stages, it should be mentioned that due to the limited time, some ideas in this section may not be implemented. Yet I will mark those functionalities which I have realized in order to make it clear.

Formulation:

- A beautiful drag & drop box for receiving image input which also has reasonable restrictions including limiting input numbers, limiting file types. ✓

- Users can preview the uploaded or uploading images in the form of thumbnails. ✔️

Initiation:

- Directly calling the backend of the platform through some APIs defined in the Flask server. ❎
- Automatically start transfering images after images were put into the dropzone.✔️
- Everytime users change the refinements, the application will automatically start the transfering process. ✔️
- Alternatives: Add a search button to start the sending process instead of auto freshing.❎
- Alternatives: Upload multiple files together and return a batch of results.❎

Review:

- Use a image gallery to provide the preview of all results. ✔️
- Support zoom effect for every images to let users inspect all results more closely. ✔️
- Set special focusing effect on each image to enhance the gallery feedback experience.✔️
- Set the distance value and sorted similarity indexes as the label of those images to provide a better insight of the result.✔️
- Users can swith the preview mode between manually clicking the next picture button and the auto gallery mode.✔️
- Users can spin(XD) the image in the galley to carefully investigate the image. ✔️
- Presenting the above information in a more fashionable way❎

Refinement:

- Users can specify their refinements:
  - Number of returning pictures: ✔️
  - What tags users want✔️ (based on relevant tags)
- User can specify the image size, image sources ❎
- Adding more images to the database and adding more metadata to all images ❎

Use:

- Users can download the images they like by clicking the individual images and click the download button in the top right corner.✔️
- Users can share the images they like also by clicking individual images and click the share button down below.✔️
- Automatically generate a comparison result for users to share their search results.❎

- Add reverse image search to provide more information about the themes of the original image.❎

Other functions:

- Calling the APIs of different search engines to get more results❎
- Using material design concepts to build a more natural and user-friendly UI✔️

About Programs

UI design

Using material design.🕶️

Frontend

- I have completely rewrite the frontend pages of this project since the original one is rather messy. They new frontend project is located in the folder 'server\static'.
- The frontend packages is constructed by webpack, however since the presenting page is rather simple, I chose to link all css, js files and js dependencies into one html(index.html). Therefore, their is no need to start the webpack services.

  For those who are still interested in how to start webpack in my project.

  - watch the modifications and build in real time.

  ```
  npm run watch
  ```

  - build the deployable version of the frontend project

  ```
  npm run build
  ```

  - just for development:

  ```
  npm run dev
  ```

- The frontend of this project use the following dependencies:
  - material web compents
  - dropzone.js
  - nanogallery2
  - jquery

Backend

- The backend modifications happen mainly in the flask server files and the search file.
- In the following part I will list out my backend modifications:
- Added a file called load_filter.py:

```
import os
import numpy as np

def getFilterDict():
    filterDict = {}
    rootDir = "./database/tags"
    for dirName, subdirList, fileList in os.walk(rootDir):
        if(dirName != rootDir):
            break
        for fname in fileList:
            if fname.find('_') == -1:
                labelName = fname.split('.')[0]
                if(labelName == "README"):
                    continue
                validArray = np.loadtxt(rootDir + "/" + fname)
                filterDict[labelName] = validArray
    return filterDict
```

- modified search logic. I will only show the part I have modified:

```
def get_all_similar(image_data, pred, pred_final, needList, number):
    acceptedId = []
    needString = needList
    needIDset = []
    for element in needString.split(','):
        needIDset.append(element)
    for word in needIDset:
        if word == "all":
            continue
        subList = filterDict[word]
        acceptedId += subList.tolist()
    acceptedId = list(map(lambda x: str(int(x)), acceptedId))
    print("total data", len(pred))
    print(image_data.shape)
    # for i in pred:
    # print(i.shape)
    # break
    os.mkdir('static/result')

    # cosine calculates the cosine distance, not similiarity. Hence no need
to reverse list
    k_ind_distance = [cosine(image_data, pred_row) \
```

```
                        for ith_row, pred_row in enumerate(pred)]
    k_ind_distance = list(filter(lambda x: x != 0, k_ind_distance))
    top_k_ind = np.argsort(k_ind_distance)
    # print(top_k_ind)
    imgNumber = 0
    correspondingDict = {}
    print(len(top_k_ind), len(pred_final))
    for i, neighbor in enumerate(top_k_ind):
        baseleftId = "image"
        name = pred_final[neighbor]
        tokens = name.split("/")
        img_name = tokens[-1]
        if imgNumber == number:
            break
        id = img_name.split('.')[0]
        id = id[2:]
        if id in acceptedId:
            image = ndimage.imread(pred_final[neighbor])
            # timestr = datetime.now().strftime("%Y%m%d%H%M%S")
            # name= timestr+"."+str(i)
            print(img_name)
            name = 'static/result/' + img_name
            correspondingDict[name] = [str(imgNumber + 1) + ':  ',
k_ind_distance[neighbor]]
            imsave(name, image)
            imgNumber += 1
    with open('recom_distance.pickle', 'wb') as f:
        pickle.dump(correspondingDict, f)
    print(imgNumber)

def get_top_k_similar(image_data, pred, pred_final, k):
        print("total data",len(pred))
        print(image_data.shape)
        #for i in pred:
        #print(i.shape)
                #break
        os.mkdir('static/result')

    # cosine calculates the cosine distance, not similiarity. Hence no need
to reverse list
        k_ind_distance = [cosine(image_data, pred_row) \
                          for ith_row, pred_row in enumerate(pred)]
        k_ind_distance = list(filter(lambda x: x != 0, k_ind_distance))
        top_k_ind = np.argsort(k_ind_distance)[:k]
        print(top_k_ind)

        correspondingDict = {}
        for i, neighbor in enumerate(top_k_ind):
```

```
            image = ndimage.imread(pred_final[neighbor])
            #timestr = datetime.now().strftime("%Y%m%d%H%M%S")
            #name= timestr+"."+str(i)
            name = pred_final[neighbor]
            tokens = name.split("/")
            img_name = tokens[-1]
            print(img_name)
            name = 'static/result/'+img_name
            correspondingDict[name] = [str(i) + ':  ',
k_ind_distance[neighbor]]
            imsave(name, image)
        with open('recom_distance.pickle', 'wb') as f:
            pickle.dump(correspondingDict, f)

            def recommend(imagePath, extracted_features, number):
        tf.reset_default_graph()

        config = tf.ConfigProto(
            device_count = {'GPU': 0}
        )

        sess = tf.Session(config=config)
        graph, bottleneck_tensor, jpeg_data_tensor, resized_image_tensor =
(create_inception_graph())
        image_data = gfile.FastGFile(imagePath, 'rb').read()
        features = run_bottleneck_on_image(sess, image_data,
jpeg_data_tensor, bottleneck_tensor)

        with open('neighbor_list_recom.pickle','rb') as f:
            neighbor_list = pickle.load(f)
        print("loaded images")
        get_top_k_similar(features, extracted_features, neighbor_list,
k=number)


def recommend_with_filter(imagePath, extracted_features, needList, number):
    tf.reset_default_graph()

    config = tf.ConfigProto(
        device_count={'GPU': 0}
    )

    sess = tf.Session(config=config)
    graph, bottleneck_tensor, jpeg_data_tensor, resized_image_tensor =
(create_inception_graph())
    image_data = gfile.FastGFile(imagePath, 'rb').read()
    features = run_bottleneck_on_image(sess, image_data, jpeg_data_tensor,
bottleneck_tensor)
```

```
    with open('neighbor_list_recom.pickle', 'rb') as f:
        neighbor_list = pickle.load(f)
    print("loaded images")
    get_all_similar(features, extracted_features, neighbor_list, needList,
number)
```

- modified the rest-server file:

```
def iter_files(rootDir):
    all_files = []
    for root, dirs, files in os.walk(rootDir):
        for file in files:
            file_name = os.path.join(root, file)
            all_files.append(file_name)
        for dirname in dirs:
            iter_files(dirname)
    return all_files

img_files = iter_files('database/dataset')
num_images = len(img_files)

@app.route('/imgUpload', methods=['GET', 'POST'])
# def allowed_file(filename):
#     return '.' in filename and \
#            filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

def upload_img():
    print("image upload")
    result = 'static/result'
    if not gfile.Exists(result):
        os.mkdir(result)
    shutil.rmtree(result)

    if request.method == 'POST' or request.method == 'GET':
        print(request.method)
        print(request.form.get("filterParam"))

        # check if the post request has the file part
        if 'file' not in request.files:
            print('No file part')
            return redirect(request.url)


        if request.form.get("filterParam") == None or
len(request.form.get("filterParam")) == 0:
            noFilter = True
        else:
            noFilter = False
```

```
        file = request.files['file']
        print(file.filename)
        # if user does not select file, browser also
        # submit a empty part without filename
        if file.filename == '' or request.form.get("number") == None:
            print('No selected file')
            return redirect(request.url)
        images = {}
        if file:  # and allowed_file(file.filename):
            filename = secure_filename(file.filename)
            file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
            inputloc = os.path.join(app.config['UPLOAD_FOLDER'], filename)
            needList = request.form.get("filterParam")
            number = request.form.get("number")
            print(number)
            if needList.find("all") != -1:
                noFilter = True
            if (noFilter):
                recommend(inputloc, extracted_features, int(number))
            else:
                needList = request.form.get("filterParam")
                recommend_with_filter(inputloc, extracted_features,
needList, int(number))
            # os.remove(inputloc)
            image_path = "/static/result"
            image_list = [os.path.join(image_path, file) for file in
os.listdir(result)
                          if not file.startswith('.')]

                # for(image in image_list):
            baseId = 'image'
            nowNum = 0
            print(len(image_list))
            while nowNum < int(number):
                images[baseId + str(nowNum)] = image_list[nowNum]
                nowNum += 1

            with open('recom_distance.pickle', 'rb') as f:
                recom_distance = pickle.load(f)
            images['recom_distance'] = recom_distance
        return jsonify(images)
```

Screenshots

My Image Search for HCI2

How many?

- All
- Sky
- Clouds
- Water
- Sea
- River
- Lake
- People
- Portrait
- Male
- Female
- Baby

"Please upload your image :>"

My Image Search for HCI2

How many?

- All
- Sky
- Clouds
- Water
- Sea
- River
- Lake
- People
- Portrait
- Male
- Female
- Baby

My Image Search for HCI2

How many?

- All
- Sky
- Clouds
- Water
- Sea
- River
- Lake
- People
- Portrait
- Male
- Female
- Baby