基于HSV颜色直方图特征和 Flask框架构建的 Web图像检索系统

机器视觉课程设计



2023

目录

[一、引言 2](#_Toc137550784)

[二、系统功能描述 2](#_Toc137550785)

[三、系统设计 3](#_Toc137550786)

[3.1 系统框图： 3](#_Toc137550787)

[3.2 组成模块任务描述： 3](#_Toc137550788)

[四、核心算法设计 4](#_Toc137550789)

[4.1 特征提取 4](#_Toc137550790)

[4.2 相似度计算 7](#_Toc137550791)

[五、系统实现 7](#_Toc137550792)

[5.1 数据集（INRIA Holidays Dataset) 7](#_Toc137550793)

[5.2 项目文件结构 8](#_Toc137550794)

[5.3 各模块介绍以及代码实现 8](#_Toc137550795)

[六、实验过程及结果 20](#_Toc137550796)

[6.1 索引文件index.csv的生成 20](#_Toc137550797)

[6.2 前端Web界面以及系统工作流程介绍 22](#_Toc137550798)

[七、算法评价及总结 26](#_Toc137550799)

[7.1 系统优势 26](#_Toc137550800)

[7.2 系统缺点 26](#_Toc137550801)

[7.3 总结 26](#_Toc137550802)

‘

## 一、引言

随着数字图像的快速增长和广泛应用，开发一个高效的图像检索系统对于快速定位和管理图像资源变得越来越重要。本项目旨在构建一套基于HSV颜色直方图特征和Flask框架构建的Web图像检索系统，用户可以通过界面选择要查询的图像，并获取数据库中其他包含相同目标的图像。

## 二、系统功能描述

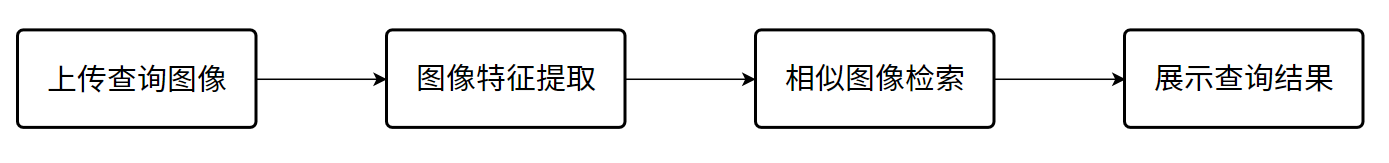
本项目的目标是开发一个准确且高效的图像检索系统，旨在满足用户的需求。该系统将提供以下功能：：

* 图像上传：用户可以通过直观友好的Web界面轻松上传查询图像。这个功能使得用户能够以简单且便捷的方式将他们感兴趣的图像添加到系统中，作为后续相似度比较的基准。
* 图像特征提取：系统将使用特定的图像特征提取算法，对上传的查询图像和数据库中的图像进行特征向量的提取。通过这一步骤，系统能够捕捉图像的重要特征，为后续的相似度计算提供基础。
* 数据库管理：项目将建立一个庞大的图像数据库，其中包含丰富的图像数据集。为了实现高效的图像索引和存储，系统将采用有效的数据管理技术，确保图像数据的快速访问和高效的存储。
* 相似图像检索：该系统能够基于查询图像的特征，快速检索数据库中与之相似度最高的相关图像。通过比较特征向量之间的相似度，系统能够快速找到最相关的图像结果，从而满足用户的检索需求。
* 用户界面：为了提供直观友好的用户体验，该系统将提供一个易于使用的用户界面。这个界面将包括图像上传功能，使用户能够方便地上传他们感兴趣的图像。此外，系统还会展示检索结果，以直观的方式呈现相似度最高的相关图像，让用户能够方便地浏览和选择感兴趣的图像。

通过实现以上功能，该图像检索系统旨在为用户提供一个可靠、高效的工具，帮助他们快速准确地搜索和检索图像数据。无论是从专业领域的研究人员、设计师，还是普通用户寻找特定图像的需求，该系统都将为他们提供便利和价值。无论是在学术研究、艺术创作、产品设计或娱乐媒体等领域，这个图像检索系统将成为一个强大的工具，促进图像资源的探索和利用。

## 三、系统设计

### 3.1 系统框图：



### 3.2 组成模块任务描述：

**用户界面模块**

用户界面模块负责提供直观友好的用户界面，以便用户能够轻松上传查询图像并获得检索结果。该模块将包括以下任务：

* 设计和开发Web界面，提供图像上传功能，让用户能够方便地上传自己的查询图像。
* 处理用户上传的图像，将其传递给图像特征提取模块进行后续处理。
* 展示检索结果，以直观的方式呈现相似度最高的相关图像，使用户能够方便地浏览和选择感兴趣的图像。

**图像特征提取模块**

图像特征提取模块的任务是使用特定的图像特征提取算法，对上传的查询图像和数据库中的图像进行特征向量的提取。该模块将执行以下任务：

* 针对查询图像和数据库中的每个图像，运行特定的特征提取算法，以提取其特征向量。
* 确保所选的特征提取算法能够捕捉到图像的重要特征，以便后续的相似度计算能够准确度量图像之间的相似程度。

**数据管理模块**

数据管理模块将负责有效地管理图像数据集和特征向量的存储，并提供图像索引和查询功能。该模块的任务包括：

* 设计和创建一个可扩展的数据库，用于存储大规模的图像数据集、特征向量和相关信息。
* 开发数据管理功能，以支持高效的图像索引和存储，确保图像数据的快速访问和检索。
* 实现图像查询功能，使得可以根据特征向量进行快速检索，找到与查询图像最相似的相关图像。

**相似图像检索模块**

相似图像检索模块是实现系统核心功能的模块，它能根据查询图像的特征向量和数据库中的特征向量计算相似度，并返回相似度最高的相关图像。该模块将完成以下任务：

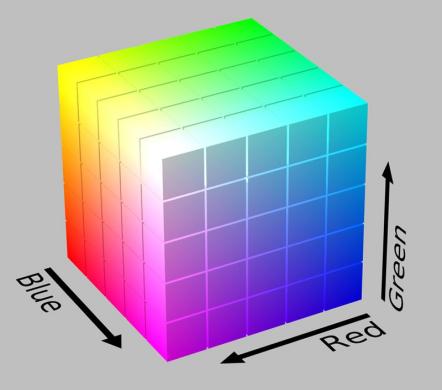
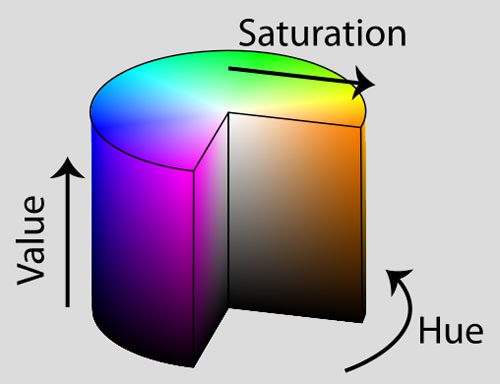
* 开发相似度计算算法，能够比较查询图像的特征向量与数据库中的特征向量，从而确定它们之间的相似度。
* 根据相似度排序，找到相似度最高的图像结果，并将其返回给用户界面模块进行展示。

以上四个模块的协同工作将使得整个图像检索系统能够高效地处理用户上传的查询图像，并快速地检索并返回与之相似度最高的相关图像。每个模块都具备特定的功能和任务，共同构成了一个完整而强大的图像检索系统。

## 四、核心算法设计

### 4.1 特征提取

**本系统采用HSV颜色直方图来描述图像特征。**

* 1. **选择色彩空间**
  + 通常，图像表示为(RGB)的三元组，我们通常将RGB色彩空间视为立方体，如下所示：
  + 
  + 然而，虽然RGB值很容易理解，但是它无法模仿人类感知颜色的方式。因此本系统使用HSV色彩空间（色调、饱和度、值）的3D颜色直方图来定义图像描述符，它将像素强度映射到圆柱体中，如下所示：
  + 
  + 本系统使用HSV色彩空间的颜色直方图作为图像特征提取算法，提取图像的颜色分布信息。颜色直方图是一种全局特征描述符，它具有对图像进行缩放和旋转而特征向量不变的性质，是一种简单而高效的特征向量表示。
  1. **选择颜色直方图bin的个数**
  + 对于HSV颜色直方图而言，过少的bin会导致查询结果的不准确，而过多的bin会导致系统失去泛化能力。因此选择合适的bin的数量极其重要。经过测试，最终选择8个bin用于色调通道，12个bin用于饱和度通道，3个bin用于value通道。最终一个图像区域HSV颜色直方图所对应的特征向量维度为8\*12\*3=288。
  1. **分割图像区域并计算颜色直方图**
  + 以整个图像的3D HSV颜色直方图作为特征有着一定的局限性，以下面的图片为例：
  + 
  + 该图包含图像顶部的蓝天和底部的沙滩，如果使用全局直方图，我们只能知道图像中存在着一定百分比的蓝色和一定百分比的棕色，而无法确定它们出现的位置。
  + 为了解决这个问题，我们可以将图像分为五个区域（左上角、左下角、右上角、右下角、中心），以模仿一种粗略的定位形式，如下图所示：
  + 
  + 这样我们就可以定位出图像中蓝色天空和棕色沙子的位置。
  + 在代码实现上，我们可以引入mask的概念，用来指示OpenCV提取何处的颜色直方图，如下图所示：
  + 
  + 只有当图像中对应位置的像素值为255（白色）的部分才会包含在直方图计算中，而像素值为0（黑色）的部分则不会包含。这样我们就实现了分别计算不同区域直方图的功能。
  + 通过该方法将图像分成五个区域，每个区域由一个直方图表示（包含8\*12\*3=288个bin），所以每张图片的整体特征向量维数为288\*5=1440。
  1. **对颜色直方图归一化**
  + 使得每个直方图都由特定条柱的相对百分比计数表示而非整数计数表示。这一步是为了确保具有相似内容但尺寸不同的图片仍然可以被成功查询（即尺度不变性）。

### 4.2 相似度计算

在本项目中，选择使用卡方距离（chi-squared distance）作为相似度的度量方法。卡方距离是一种常用的非负度量，用于比较两个向量之间的相似程度。对于图像特征向量的比较，卡方距离的计算公式如下：

卡方距离(D) = sum((Xi - Yi)^2 / (Xi + Yi))

其中，Xi和Yi分别表示两个特征向量中对应维度的元素值。

卡方距离的计算过程如下：

1. 对于查询图像的特征向量和数据库中每个图像的特征向量，分别计算每个维度的差值（Xi - Yi）。
2. 对于每个维度的差值，将其平方并除以对应维度的总和（Xi + Yi）。
3. 对所有维度的差值平方除以总和的结果进行累加，得到卡方距离。

卡方距离的值越小，表示两个图像的特征向量越相似，即图像之间的相似度越高。

在相似图像检索模块中，将使用卡方距离来计算查询图像的特征向量与数据库中各个图像的特征向量之间的相似度。通过对数据库中的所有图像进行计算和比较，系统将能够确定与查询图像最相似的图像，并按照相似度的大小进行排序，以便返回相似度最高的相关图像给用户。

选择卡方距离作为相似度度量方法的原因是它能够捕捉到特征向量之间的差异，并且在图像检索任务中表现良好。通过使用卡方距离作为度量方法，我们能够准确地衡量图像之间的相似程度，并为用户提供准确、可靠的图像检索结果。

## 五、系统实现

### 5.1 数据集（INRIA Holidays Dataset)

数据集INRIA Holidays Dataset是一个用于评估图像搜索的数据集，由法国国家信息与自动化研究所（INRIA）创建。该数据集主要包含一些作者的个人假期照片，涵盖了各种不同的场景和对象，如埃及金字塔、海滩、水下潜水等。数据集中有500个图像组，每个组代表一个不同的场景或对象。每个组的第一张图像是查询图像，其他图像是正确的检索结果。数据集中共有1491张高分辨率的图像，以及4455091个128维的SIFT特征描述符。下载地址：[Download datasets (inrialpes.fr)](http://lear.inrialpes.fr/people/jegou/data.php)

**注意：由于文件大小限制，提交上去的压缩包中没有包含该数据集内容，如果想要运行代码，请将下载下来的数据集存放到/app/static/dataset中。**

下面是数据集图片的一些示例：



### 5.2 项目文件结构

├── app

│   ├── app.py

│   ├── index.py

│   ├── index.csv

│   ├── pyimagesearch

│   │   ├── FeatureDescriptor.py

│   │   └── searcher.py

│   ├── static

│   │   ├── main.css

│   │   └── main.js

│   │   └── queries

│   │   └── dataset

│   └── templates

│       ├── \_base.html

│       └── index.html

├── config

│   ├── app.ini

│   ├── nginx.conf

│   └── supervisor.conf

└── requirements.txt

### 5.3 各模块介绍以及代码实现

**FeatureDescriptor.py:**

定义了一个名为FeatureDescriptor的类，该类包含:

1. \_init\_ 方法：初始化参数（直方图bin的个数）。
2. describe方法：该方法接受一个图像并返回该图像的HSV颜色直方图特征。

具体实现为：①将图像转换为HSV颜色空间。

②将图像分为五个区域（左上角、左下角、右上角、右下角、中 心），以模仿一种粗略的定位形式。

③遍历5个区域，提取5个区域的HSV颜色直方图特征。

④将这些特征合并为一个特征向量。

1. histogram方法：具体实现计算图像某一区域的HSV颜色直方图特征，并归一化使得该特征具有尺度不变性。

代码实现如下：

1. import numpy as np
2. import cv2
3. *# 提取图像的3D HSV颜色直方图*
4. class FeatureDescriptor:
5. def \_\_init\_\_(self, bins):
6. *# 存放直方图的bin的个数*
7. self.bins = bins
8. *# 将RGB转换为HSV并初始化特征以量化和表示图像*
9. def describe(self, image):
10. image = cv2.cvtColor(image, cv2.COLOR\_BGR2HSV)
11. features = []
12. *# 获取图像的尺寸并计算图像的中心*
13. (h, w) = image.shape[:2]
14. (cx, cy) = (int(w \* 0.5), int(h \* 0.5))
15. *# 将图像分成四个部分（左上角，右上角，右下角，左下角）*
16. segments = [(0, cx, 0, cy), (0, cx, cy, h), (cx, w, cy, h), (cx, w, 0, cy)]
17. *# 构造一个椭圆形的mask，代表图像的中心，大小为图像的0.75倍*
18. (axesX, axesY) = (int(int(w \* 0.75) / 2), int(int(h \* 0.75) / 2))
19. ellipse\_mask = np.zeros(image.shape[:2], dtype="uint8")
20. cv2.ellipse(ellipse\_mask, (cx, cy), (axesX, axesY), 0, 0, 360, 255, -1)
21. *# 遍历四个corner*
22. for seg in segments:
23. *# 构造每个corner的mask*
24. corner\_mask = np.zeros(image.shape[:2], dtype='uint8')
25. corner\_mask[seg[0]:seg[1], seg[2]:seg[3]] = 255
26. corner\_mask = cv2.subtract(corner\_mask, ellipse\_mask)
27. *# 提取对应corner区域的HSV直方图特征*
28. hist = self.histogram(image, corner\_mask)
29. features.extend(hist)
30. *# 提取中心椭圆区域的HSV直方图特征*
31. hist\_ellipse = self.histogram(image, ellipse\_mask)
32. features.extend(hist\_ellipse)
33. return features
34. *# 计算图像的对应mask区域的直方图*
35. def histogram(self, image, mask):
36. hist = cv2.calcHist([image], [0, 1, 2], mask, self.bins,
37. [0, 180, 0, 256, 0, 256])
38. *# 归一化，使得直方图具有尺度不变性*
39. hist = cv2.normalize(hist, hist).flatten()
40. return hist

**index.py:**

使用了前面实现的FeatureDescriptor模块，用于提取数据集中每个图片的HSV颜色直方图特征，并将这些特征写入索引文件index.csv中。

具体实现为：①首先创建一个名为cd的FeatureDescriptor对象，该对象使用 8个色调bin，12个饱和度bin和3个value bin来描述图像每 个区域的颜色特征。

②使用glob模块遍历数据集中的所有图像，对于每个图像，使 用cv2.imread()方法读取图像，并使用cd.describe()方法提取 图像的特征。该方法将图像分成五个区域，每个区域由一个直 方图表示（包含 8\*12\*3=288个bin），所以每张图片的整体 特征向量维数为288\*5=1440。

③最后，将图像的文件名和对应的特征向量写入索引文件 index.csv中，并输出索引文件。

代码实现如下：

1. from pyimagesearch.FeatureDescriptor import FeatureDescriptor
2. import argparse
3. import glob
4. import cv2
5. *# 构造参数解析器并解析参数*
6. *#  python index.py --dataset static/dataset --index index.csv*
7. ap = argparse.ArgumentParser()
8. *# --dataset: 数据集的路径*
9. ap.add\_argument("-d", "--dataset", required = True,
10. help = "Path to the directory that contains the images to be indexed")
11. *# --index: 从数据集中提取的图像特征index.csv文件的路径*
12. ap.add\_argument("-i", "--index", required = True,
13. help = "Path to where the computed index will be stored")
14. args = vars(ap.parse\_args())
15. *# 初始化FeatureDescriptor类*
16. *# 8个色调bin，12个饱和度bin，3个value bin*
17. cd = FeatureDescriptor((8, 12, 3))
18. *# 图像特征索引文件output*
19. output = open(args["index"], "w")
20. *# 遍历数据集中的所有图片*
21. for imagePath in glob.glob(args["dataset"] + "/\*.jpg"):
22. *# imageID：图像的唯一文件名*
23. imageID = imagePath.split("\\")[-1]
24. image = cv2.imread(imagePath)
25. *# 提取图像的特征并写入文件*
26. *# 图像分为五个区域，每个区域由一个直方图表示（包含8\*12\*3=288个bin），所以整体特征向量维数为288\*5=1440*
27. features = cd.describe(image)
28. features = [str(f) for f in features]
29. output.write("%s,%s\n" % (imageID, ",".join(features)))
30. output.close()

**searcher.py:**

定义了一个名为Searcher的类，该类包含:

1. \_init\_ 方法：初始化参数（特征索引文件index.csv的路径）。
2. search方法：该方法接受一个特征向量queryFeatures和一个卡方距离门限limit，然后在索引文件中搜索与queryFeatures最相似的图像，并返回搜索结果。

具体实现为：①使用csv模块打开索引文件，遍历文件中的每一行，提取每个 图像的特征向量。

②计算queryFeatures和该特征向量之间的卡方距离。

③如果卡方距离小于等于limit，则将该图像的文件名和卡方距 离存储在一个字典中。

④按照卡方距离从小到大排序，并返回排序后的字典。

1. chi2\_distance方法：具体实现计算两个特征向量之间的卡方距离。

代码实现如下：

1. import numpy as np
2. import csv
3. class Searcher:
4. def \_\_init\_\_(self, indexPath):
5. self.indexPath = indexPath
6. # 寻找最相似的图像。queryFeatures: 从待检索图像中提取的特征,limit:卡方距离门限，当d<=limit时，认为两幅图像相似
7. def search(self, queryFeatures,limit = 6.8):
8. *# results存放检索结果*
9. results = {}
10. with open(self.indexPath) as f:
11. reader = csv.reader(f)
12. for row in reader:
13. *# 提取index.csv文件中存储的图像名和特征向量。row[0]: 图像的唯一文件名，row[1:]: 图像的特征向量*
14. features = [float(x) for x in row[1:]]
15. *# 计算待检索图像和index.csv文件中图像的特征向量的卡方距离*
16. d = self.chi2\_distance(features, queryFeatures)
17. *# 当卡方距离小于等于limit时，将卡方距离和图像名存入results中*
18. if d <= limit:
19. results[row[0]] = d
20. f.close()
21. *# 按照卡方距离从小到大排序（卡方距离越小，越相似）*
22. results = sorted([(v, k) for (k, v) in results.items()])
23. if len(results) >= 6:
24. results = results[:6]
25. return results
26. # 计算两个特征向量的卡方距离
27. def chi2\_distance(self, histA, histB, eps = 1e-10):
28. d = 0.5 \* np.sum([((a - b) \*\* 2) / (a + b + eps)
29. for (a, b) in zip(histA, histB)])
30. return d

**app.py:**

搭建了一个Flask框架并实现了Web界面后端的图像检索功能，具体为：

1. 创建了一个名为app的Flask实例，并使用static\_url\_path和static\_folder参数指定了静态文件的URL路径和文件夹路径。
2. 定义了三个route：'/','/upload'和’/search’。

①'/'作为main route，返回一个HTML模板。

②'/upload' route 接收POST请求，从请求中获取查询图像的本地路径。

③'/search' route 接收POST请求，从请求中获取查询图像的本地路径，并借此访问查询图像、提取特征、查找相似图像并返回结果，具体流程如下：

1. 使用前面实现的FeatureDescriptor模块提取图像的HSV颜色直方图特征。
2. 使用前面实现的Searcher模块在索引文件index.csv中搜索相似的图像。
3. 最后将搜索结果作为JSON响应返回给客户端:

* 若检索到相似图像，返回存放图片名称image和相似度（卡方距离）score的字典RESULTS\_ARRAY。
* 若没有检索到相似图像或出现其他错误，返回jsonify({"sorry":"Sorry, no results! Please try again."}), 500。

代码实现如下：

1. import os
2. import cv2
3. from flask import Flask, render\_template, request, jsonify
4. from pyimagesearch.FeatureDescriptor import FeatureDescriptor
5. from pyimagesearch.searcher import Searcher
6. *# 创建Flask实例，指定静态文件夹为static*
7. app = Flask(\_\_name\_\_, static\_url\_path = "", static\_folder = "static")
8. *# 索引文件index.csv的路径*
9. INDEX = os.path.join(os.path.dirname(\_\_file\_\_), 'index.csv')
10. *# main route*
11. @app.route('/')
12. def index():
13. return render\_template('index.html')
14. *# upload route*
15. @app.route('/upload', methods=['POST'])
16. def upload():
17. if request.method == "POST":
18. file=request.files['image']
19. *# get path*
20. image\_path='queries/'+file.filename
21. *# 返回查询图像的路径*
22. return jsonify({'image\_path': image\_path})
23. *# search route*
24. @app.route('/search', methods=['POST'])
25. def search():
26. if request.method == "POST":
27. file=request.files['image']
28. RESULTS\_ARRAY = []
29. *# get path*
30. image\_path='D:\\cv\_design\\image\_search\\app\\static\\queries\\'+file.filename
31. *# 输出查询图像的路径*
32. print(image\_path)
33. try:
34. *# 初始化FeatureDescriptor类*
35. *# 8个色调bin，12个饱和度bin，3个value bin*
36. cd = FeatureDescriptor((8, 12, 3))
37. *# 读取查询图像*
38. query = cv2.imread(image\_path)
39. *# 提取查询图像的特征*
40. features = cd.describe(query)
41. *# 使用Searcher类在索引文件中查询相似图像*
42. searcher = Searcher(INDEX)
43. results = searcher.search(features)
44. *# 将检索结果存入RESULTS\_ARRAY中，包含图像名和相似度*
45. for (score, resultID) in results:
46. RESULTS\_ARRAY.append(
47. {"image": str(resultID), "score": str(score)})
48. *# return success*
49. if len(RESULTS\_ARRAY) == 0:
50. return jsonify({"sorry": "Sorry, no results! Please try again."}), 500
51. else:
52. return jsonify(results=RESULTS\_ARRAY)
53. except:
54. *# return error*
55. return jsonify({"sorry": "Sorry, no results! Please try again."}), 500
56. *# run!*
57. if \_\_name\_\_ == '\_\_main\_\_':
58. app.run('0.0.0.0', debug=True)

**main.js:**

实现Web页面的前端页面逻辑，它的作用是在页面加载完成后，为“上传图像”的button添加点击事件处理程序。当用户上传了要查询的图像后，上传的图像将通过POST请求发送到后端Flask应用程序（app.py）进行处理。一旦处理完成，上传的图像和查询结果将显示在前端Web页面上。

代码实现如下：

1. *// ----- custom js ----- //*
2. *// hide initial*
3. $("#searching").hide();
4. $("#results-table").hide();
5. $("#error").hide();
6. *// 全局变量*
7. var url = 'dataset/';
8. var data = [];
9. $(function () {
10. *// sanity check*
11. console.log("ready!");
12. *// 获取上传图片按钮*
13. var uploadBtn = document.querySelector('button');
14. *// 添加点击事件监听器*
15. uploadBtn.addEventListener('click', function () {
16. *// 创建input元素*
17. var input = document.createElement('input');
18. input.type = 'file';
19. *// 添加change事件监听器*
20. input.addEventListener('change', function () {
21. *// 获取上传的图片文件*
22. var file = input.files[0];
23. console.log(file);
24. *// 创建FormData对象*
25. var formData = new FormData();
26. formData.append('image', file);
27. *// empty/hide results*
28. $("#results").empty();
29. $("#results-table").hide();
30. $("#error").hide();
31. *// show searching*
32. $("#searching").show();
33. console.log("searching...");
34. $.ajax({
35. type: "POST",
36. url: "/upload",
37. data: formData,
38. processData: false,
39. contentType: false,
40. success: function (result) {
41. *// 获取显示用户上传的图像的img元素*
42. var uploadedImage = document.getElementById('uploaded-image');
43. *// 获取用户上传的图像的路径*
44. var imagePath = result['image\_path'];
45. *// 使用url\_for函数来生成静态文件的URL*
46. var imageUrl = imagePath;
47. *// 设置img元素的为用户上传的图像*
48. uploadedImage.src = imageUrl;
49. console.log(imageUrl);
50. *// 设置img元素样式*
51. uploadedImage.style.height = '300px'
52. uploadedImage.style.border = '1px solid #ccc';
53. uploadedImage.style.borderRadius = '5px';
54. uploadedImage.style.boxShadow = '2px 2px 5px rgba(0, 0, 0, 0.3)';
55. uploadedImage.style.padding = '10px';
56. },
57. error: function (error) {
58. console.log(error);
59. }
60. })
61. *// 发送POST请求*
62. $.ajax({
63. type: "POST",
64. url: "/search",
65. data: formData,
66. processData: false,
67. contentType: false,
68. success: function (result) {
69. console.log(result.results);
70. var data = result.results
71. $("#searching").hide();
72. *// show result table*
73. $("#results-table").show();
74. *// 将查询结果添加到dom*
75. for (i = 0; i < data.length; i++) {
76. $("#results").append('<tr><th><a href="' + url + data[i]["image"] + '"><img src="' + url + data[i]["image"] +
77. '" class="result-img"></a></th><th>' + data[i]['score'] + '</th></tr>')
78. };
79. },
80. error: function (error) {
81. $("#searching").hide();
82. console.log(error);
83. *// 将错误信息添加到dom*
84. $("#error").append();
85. $("#error").show();
86. }
87. });
88. });
89. *// 触发input元素的点击事件*
90. input.click();
91. });
92. });

**\_base.html、index.html、main.css**三者为前端Web页面的构建，这里就仅展示代码，不再做额外说明**。**

**\_base.html:**

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="utf-8">
5. <meta http-equiv="X-UA-Compatible" content="IE=edge">
6. <meta name="viewport" content="width=device-width, initial-scale=1">
7. <meta name="description" content="">
8. <meta name="author" content="">
9. <title>Image Search</title>
10. *<!-- stylesheets -->*
11. <link href="//maxcdn.bootstrapcdn.com/bootswatch/3.2.0/yeti/bootstrap.min.css" rel="stylesheet" media="screen">
12. <link href="{{ url\_for('static', filename='main.css') }}" rel="stylesheet">
13. </head>
14. <body>
15. *<!-- Page Content -->*
16. <div class="container center-container">
17. *<!-- messages -->*
18. {% for message in get\_flashed\_messages() %}
19. <div class="alert alert-success alert-dismissible" role="alert">
20. <button type="button" class="close" data-dismiss="alert"><span aria-hidden="true">&times;</span><span class="sr-only">Close</span></button>
21. {{ message }}
22. </div>
23. {% endfor %}
24. *<!-- child template -->*
25. {% block content %}{% endblock %}
26. *<!-- errors -->*
27. {% if error %}
28. <p class="error"><strong>Error:</strong> {{ error }}</p>
29. {% endif %}
30. </div>
31. *<!-- /.container -->*
32. *<!-- Scripts -->*
33. <script src="//code.jquery.com/jquery-2.1.1.min.js" type="text/javascript"></script>
34. <script src="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/js/bootstrap.min.js" type="text/javascript"></script>
35. <script src="{{ url\_for('static', filename='main.js') }}" type="text/javascript"></script>
36. </body>
37. </html>

**index.html:**

1. {% extends "\_base.html" %}
2. {% block content %}
3. <div class="row">
4. <div class="col-md-7">
5. <h2>请上传要查询的图像</h2>
6. <button>上传图像</button>
7. <p><img id="uploaded-image"></p>
8. <br>
9. </div>
10. <div class="col-md-3 col-md-offset-1">
11. <h2>查询结果</h2>
12. <br>
13. <p id="searching">Searching...</p>
14. <p id="error">没有查询到相关图像!</p>
15. <table class="table" id="results-table">
16. <thead>
17. <tr>
18. <th>Image</th>
19. <th>Score</th>
20. </tr>
21. </thead>
22. <tbody id="results">
23. </tbody>
24. </table>
25. </div>
26. </div>
27. <br>
28. {% endblock %}

**main.css:**

1. */\* custom styles \*/*
2. .center-container {
3. text-align: center;
4. padding-top: 20px;
5. padding-bottom: 20px;
6. }
7. .result-img {
8. max-width: 100px;
9. max-height: 100px;
10. }
11. body {
12. background-color: #f0f0f0;
13. font-family: "Microsoft YaHei", sans-serif;
14. }
16. h1, h2, h3, h4, h5, h6 {
17. color: #333333;
18. }
19. button {
20. margin-top: 10px;
21. margin-bottom: 10px;
22. background-color: #4CAF50;
23. border: none;
24. color: white;
25. padding: 10px 20px;
26. text-align: center;
27. text-decoration: none;
28. display: inline-block;
29. font-size: 16px;
30. border-radius: 5px;
31. cursor: pointer;
32. transition-duration: 0.4s;
33. }
35. button:hover {
36. background-color: #3e8e41;
37. }
38. #uploaded-image {
39. margin-top: 10px;
40. }

## 六、实验过程及结果

### 6.1 索引文件index.csv的生成

回看前面index.py中参数解析器的构造部分，代码如下：

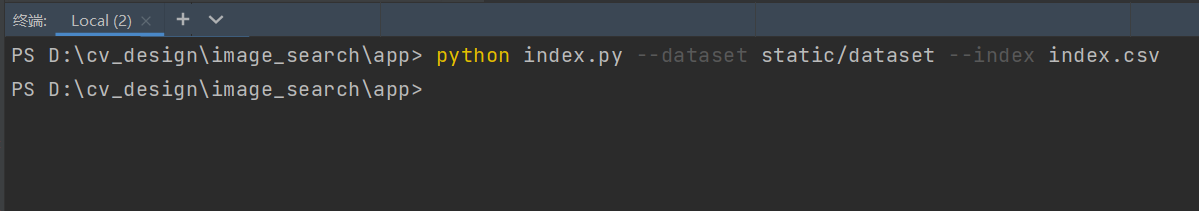
1. *# 构造参数解析器并解析参数*
2. ap = argparse.ArgumentParser()
3. *# --dataset: 数据集的路径*
4. ap.add\_argument("-d", "*--dataset", required = True,*
5. help = "Path to the directory that contains the images to be indexed")
6. *# --index: 从数据集中提取的图像特征index.csv文件的路径*
7. ap.add\_argument("-i", "--index", required = True,
8. help = "Path to where the computed index will be stored")

从上面代码可以得知，运行index.py的终端命令行格式为：

python index.py --dataset （要索引的数据集文件路径） --index （索引文件index.csv存储路径）

结合之前介绍的项目文件结构可知，数据集存储在index.py的同一级文件static下，而索引文件index.csv与index.py存储在同一级文件中，因此，要生成索引文件index.py，需在终端输入如下命令行：

python index.py --dataset static/dataset --index index.csv



由于数据库规模较大，因此索引文件index.csv的生成需要一定时间，请耐心等待。

前面介绍过，对于HSV颜色直方图而言，过少的bin会导致查询结果的不准确，而过多的bin会导致系统失去泛化能力。因此选择合适的bin的数量极其重要。经过测试，最终选择8个bin用于色调通道，12个bin用于饱和度通道，3个bin用于value通道。所以每个图像区域对应的特征向量维度为8\*12\*3=288个bin。又由于我们的算法将图像分为5个区域（左上角、左下角、右上角、右下角、中心），每个区域由一个直方图表示，所以每张图像的整体特征向量维数为288\*5=1440维。因此index.csv文件的行数为数据集中图片的个数，列数为1441列（首列为图片名字）。具体如下：

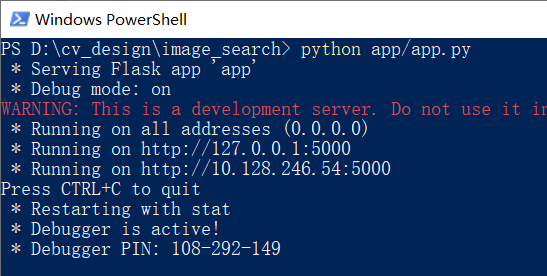
**索引文件index.csv(仅展示部分内容）:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | A | B | C | D | E | F | G | H |
| 1 | 100000.jpg | 0 | 0 | 0.99649566 | 0 | 0 | 0 | 0 |
| 2 | 100001.jpg | 1.20E-05 | 1.51E-05 | 0.9521937 | 3.01E-06 | 6.02E-06 | 0 | 1.51E-06 |
| 3 | 100002.jpg | 0.001477161 | 0 | 0.002684768 | 3.05E-05 | 0 | 0 | 5.93E-05 |
| 4 | 100100.jpg | 0.000304989 | 0.001253531 | 0.002842829 | 4.48E-05 | 0.00083662 | 0.000623967 | 0.000215451 |
| 5 | 100101.jpg | 0.006305663 | 0.005110139 | 0.001202814 | 0.001417863 | 0.001545434 | 1.82E-05 | 0.00170581 |
| 6 | 100200.jpg | 0.0007407 | 0.000204974 | 6.06E-05 | 0.000619579 | 0.000125779 | 4.66E-05 | 0.000861821 |
| 7 | 100201.jpg | 0.13232836 | 0.009261079 | 0.003337898 | 0.021058569 | 0.002990913 | 6.35E-05 | 0.011895233 |
| 8 | 100300.jpg | 3.38E-05 | 0.000115498 | 0.006183379 | 0 | 1.97E-05 | 0.000107047 | 0 |
| 9 | 100301.jpg | 0.001609946 | 0.003118503 | 4.61E-05 | 0.00100468 | 0.007788575 | 0.000967811 | 0.000417849 |
| 10 | 100302.jpg | 0.001172023 | 0.004135252 | 0.000497557 | 0.001009856 | 0.006405585 | 0.001017228 | 0.000641296 |

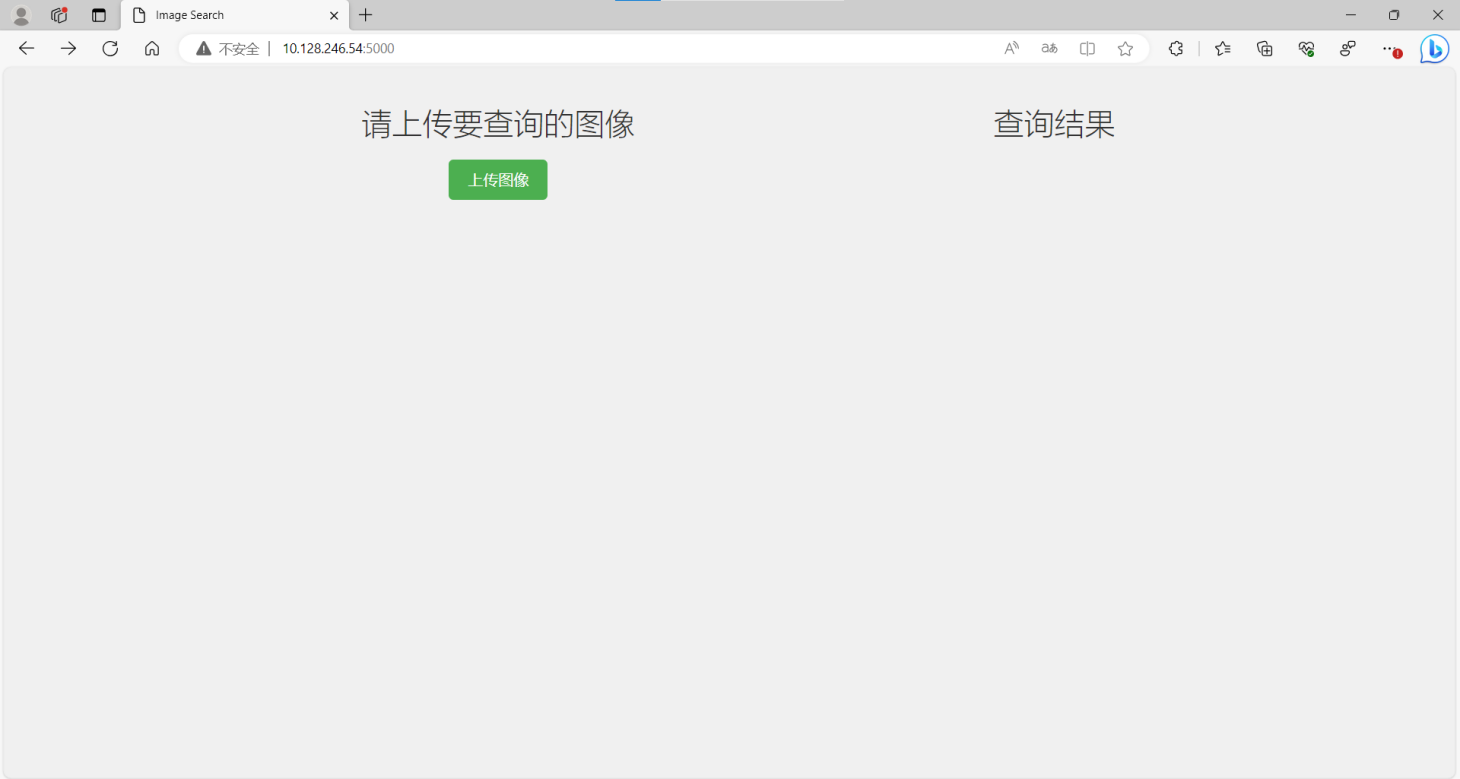
### 

### 6.2 前端Web界面以及系统工作流程介绍

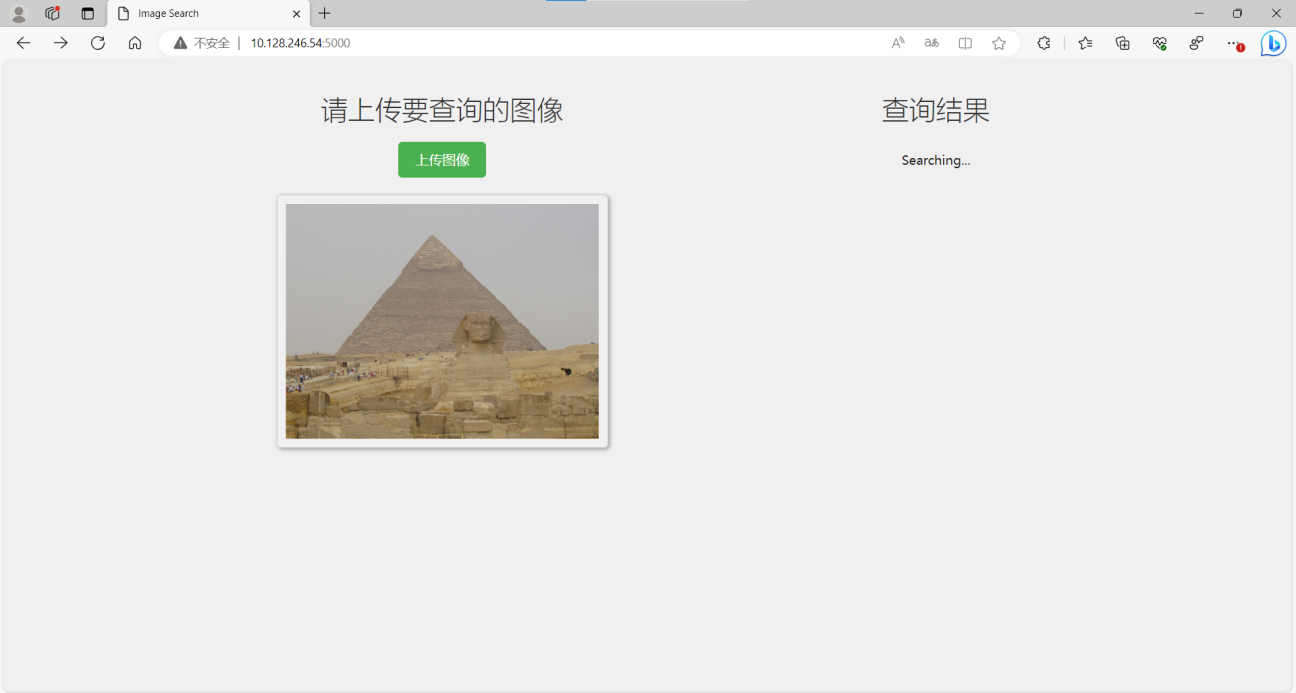
在根目录下打开终端，并输入：python app/app.py。回车执行：



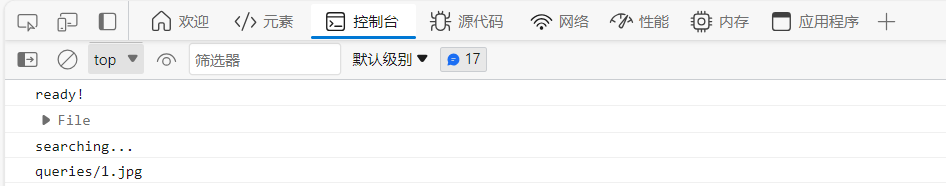
在浏览器中输入上述地址进入前端Web界面，初始页面如下图所示：



点击”上传图像”按钮上传要查询的图像，如下图所示：



可以看到Web前端界面显示出了用户上传的查询图像，右侧显示Searching...表明正在进行查询。此时触发main.js中的点击事件处理程序，main.js向后端app.py发出一个POST请求，app.py中的'/search' route 接收POST请求，从请求中获取查询图像的本地路径。控制台中输出相应信息。



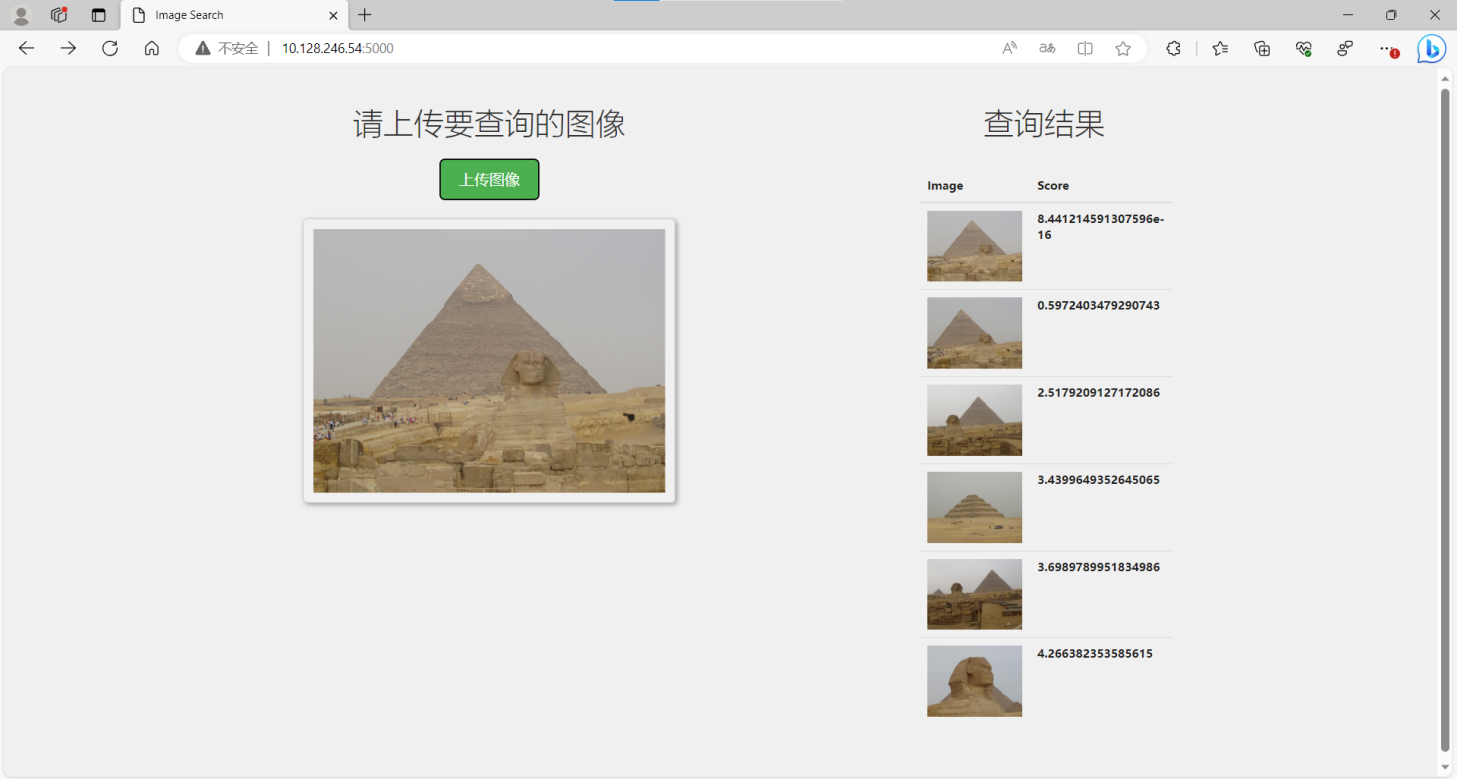
在获得查询图像的本地路径后，app.py会进行如下操作：

1. 使用FeatureDescriptor类提取图像的HSV颜色直方图特征。
2. 使用Searcher类在索引文件index.csv中搜索相似的图像。
3. 最后将搜索结果或错误信息作为JSON响应返回给客户端。

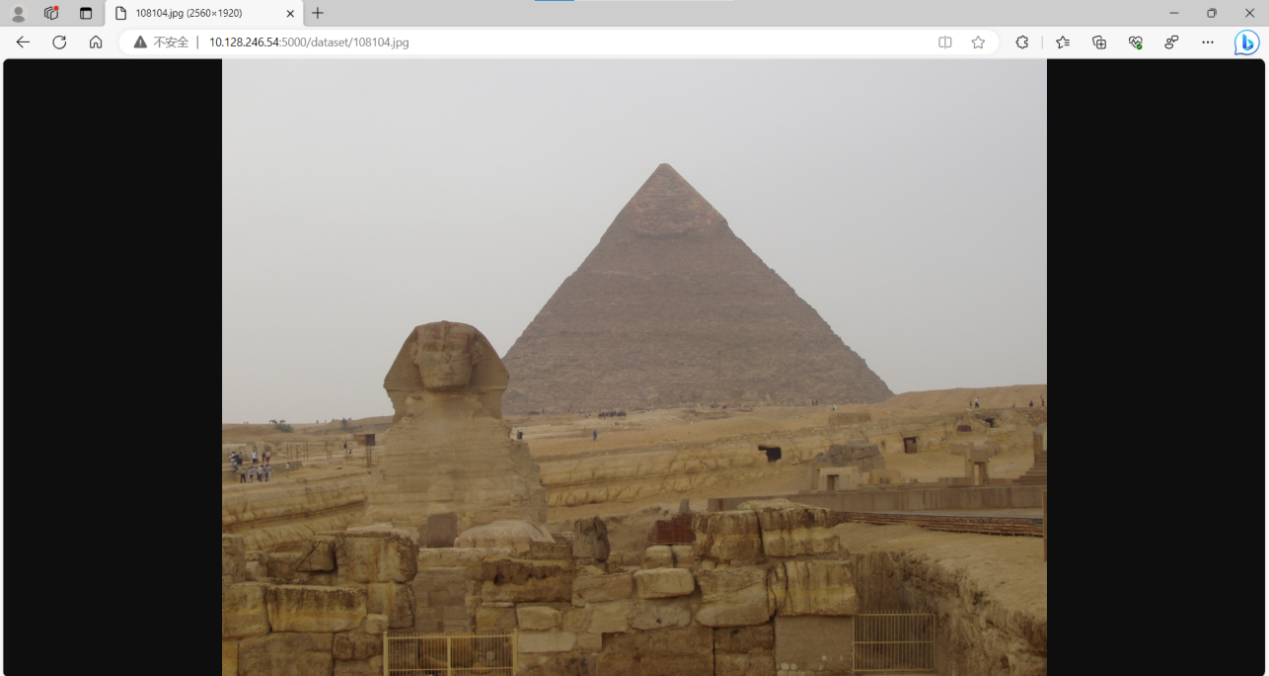
下图展示了搜索成功所返回的json响应：存放图片名称image和相似度（卡方距离）score的的字典Array。



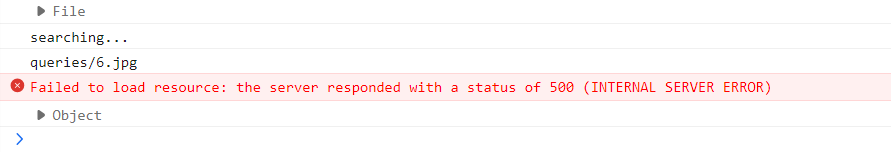
下图是搜索成功的Web前端页面展示：



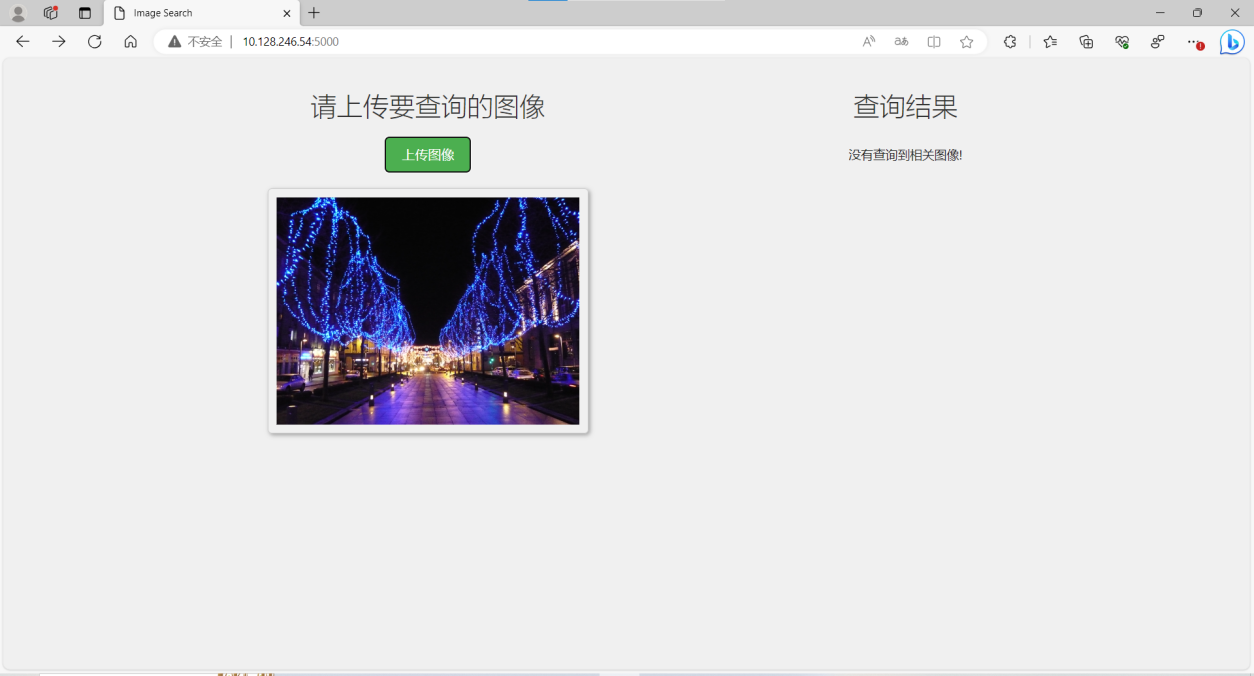
可以看到在查询结果中输出了查询到的Image以及其对应的相似度Score（卡方距离）,按相似度从大到小进行排列。点击结果图像可以看查询结果的大图。



若没有检索到相似图像或出现了其它错误，返回json响应：jsonify({"sorry": "Sorry, no results! Please try again."}), 500。如下图所示：



对应的Web前端页面如下图所示：



## 七、算法评价及总结

### 7.1 系统优势

* 特征简单高效：以HSV颜色直方图作为图像的特征表示，原理简单，特征提取也相对容易，并且具有对图像进行缩放和旋转而特征向量不变的性质，是一种简单而高效的特征向量表示形式。
* 特征有序表示：系统将图像分为五个区域（左上角、左下角、右上角、右下角、中心），以模仿一种粗略的定位形式，并引入mask来逐一提取每个部分的HSV颜色直方图特征向量。防止将颜色相似但分布截然不同（举个不恰当的例子：“爱我中国”和“我中爱国”）的两张图像判定为相似图像。
* 直观易用：系统使用Flask框架搭建了一个直观友好的Web前端界面，包括图像查询功能和检索结果的展示，相较于传统的终端命令行形式更方便用户使用。

### 7.2 系统缺点

* HSV颜色直方图特征不能很好地应对图像的光照变化（如在高光强和低光强下拍摄的同一场景由于颜色亮度不同很可能被判为不相似），也无法捕捉图像的高层语义信息。
* 该系统采用的是简单的本地CSV文件的形式用作数据集的存储和索引，而非实际的数据库形式，并且没有使用特征降维、特征聚类、反馈技术等方法来优化检索性能，因此可能会导致图像检索速度慢、内存占用大等问题。

### 7.3 总结

在机器视觉技术课程设计中，我们小组成员相互协作，共同完成了基于HSV颜色直方图特征和Flask框架构建的Web图像检索系统的设计与实现。通过完成特征提取、相似度比较、数据库索引构建、前端Web界面搭建等任务，实现了良好处理图像检索并查找数据库中相似图像的功能。

在实验的过程中，我们对图像的矩阵表示、特征提取、卷积及拟合、相似度计算等知识有了更为深刻的理解。同时在编写程序的过程中也进一步提升了我们项目搭建与团队协作的能力。

通过本学期机器视觉技术课程的学习以及课程设计的完成，团队成员对机器视觉技术有了一个较为基础、全面的认识。但要想深入理解，还需在今后的学习生涯中深入实践。周琬婷老师在这学期本课程的实践课程让我们首次接触到了比较系统化的机器视觉编程实践，为我们接下来深入理解机器视觉技术奠定了基础。我们将会在今后的学习生涯中继续开拓、不断实践。在此感谢周老师本学期带给我们的精彩的课程，感谢老师对同学们的耐心讲解与悉心教导。