Final Project

Image Search/Retrieval Application

Khalid Alabyad 201-700-500

Moaaz Al-Akel 201-600-819

INTRODUCTION

This is the Final Project for the computer vision Course CIE 552.

It is a Flask application for image search and retrieval using several methods including: SIIFT, CBIR, Pre trained deep learning model

The Project can be found here on github

The Data used in this project are : Caltch-101 and The INRIA holidays dataset

Code structure:

The application structure is :

```
ImageSearchEngine
|—server.py
|—feature_extractor.py
|—offline.py
|—static
|   ├— CbirFeature
|   ├— img
|   ├— Image
|   ├— uploaded
|   └— SiftFeature
|—templates
|   ├— index.html
|   └—indexSift.html
|—Procfile
└—requirements.txt
```

There are 3 code files:

1. feature_extractor.py which holds the classes to get features of images.

As mentioned before, we are using 2 different methods to get features of the images: SIFT, CBIR. so we have 2 classes for both descriptors.

```
import cv2
import numpy as np
import imutils



class CbirExtractor:
   def __init__(self,bins):
       self.bins=bins # number of bins
   def extract(self, img):
       """
       Extract a deep feature from an input image
       Args:
           img: a BGR from imread
           Returns:
           feature (np.ndarray): deep feature with the shape=(#p,128)
       """
       image = cv2.cvtColor(img, cv2.COLOR_BGR2HSV) # convert it to HSV
color space
       features = []
       # now getting the dimensions of the image and its center:
       (h, w) = image.shape[:2]
       (cX, cY) = (int(w * 0.5), int(h * 0.5))
       #For our image descriptor, we are going to divide our image into
five different regions:
       #(1) the top-left corner, (2) the top-right corner, (3) the
bottom-right corner,
       #(4) the bottom-left corner, and finally (5) the center of the
image.
       #first get the top-lift,top-right,bottom-lift,bottom-right
       segments = [(0, cX, 0, cY), (cX, w, 0, cY), (cX, w, cY, h),(0, cX,
cY, h)]
       # then get the center by applying an elliptical mask
       (axesX, axesY) = (int(w * 0.75) // 2, int(h * 0.75) // 2)
```

```
        ellipMask = np.zeros(image.shape[:2], dtype = "uint8")
        cv2.ellipse(ellipMask, (cX, cY), (axesX, axesY), 0, 0, 360, 255, -1)
        #----------------------
        # now we will be getting the histogram for the 5 regions
        # first get gor the 4 regions on the corners:-
        for (startX, endX, startY, endY) in segments:
            # construct a mask for each corner of the image, subtracting
            # the elliptical center from it
            cornerMask = np.zeros(image.shape[:2], dtype = "uint8")
            cv2.rectangle(cornerMask, (startX, startY), (endX, endY), 255,
-1)
            cornerMask = cv2.subtract(cornerMask, ellipMask)
            # extract a color histogram from the image, then update the
            # feature vector
            hist = self.histogram(image, cornerMask)
            features.extend(hist)
        # now for the center region
        hist = self.histogram(image, ellipMask)
        features.extend(hist)
        return features

    def histogram(self, image, mask):
        # extract a 3D color histogram from the masked region of the
        # image, using the supplied number of bins per channel
        hist = cv2.calcHist([image], [0, 1, 2], mask, self.bins,[0, 180, 0,
256, 0, 256])
        # normalize the histogram if we are using OpenCV 2.4

        hist = cv2.normalize(hist, hist).flatten()
        # return the histogram
        return hist


    def chi2_distance(self,A, B):
        eps = 1e-10 # add it to prevent dividing by zero
        # compute the chi-squared distance from
https://www.geeksforgeeks.org/chi-square-distance-in-python/
        chi = 0.5 * np.sum([((a - b) ** 2) / (a + b+eps) for (a, b) in
zip(A, B)])
        return chi
```

```
class SiftExtractor:
    def __init__(self):
        self.model=cv2.xfeatures2d.SIFT_create()
        #self.FLANN_INDEX_KDTREE=1
        self.index_params=dict(algorithm=1,trees=5)
        self.search_params=dict(checks=50)   # or pass empty dictionary

self.flann=cv2.FlannBasedMatcher(self.index_params,self.search_params)
    def extract(self, img):
        """
        Extract a deep feature from an input image
        Args:
            img: a gray scale image from cv2.imread
            Returns:
            feature (np.ndarray): deep feature with the shape=(#p,128)
        """
        _,feature = self.model.detectAndCompute(img,None)  # (1, 4096) ->
(4096, )
        return feature
    def matching(self,des1,des2):
        matches=self.flann.knnMatch(des1,des2,k=2)
        N_matches=sum([1 for m,n in matches if m.distance < 0.5*n.distance])
        if N_matches == 0:
            return 1000
        else:
            return 1.0/N_matches*1000
```

2.offline.py which applies the features extraction over all the images in the database

```
from PIL import Image
from feature_extractor import CbirExtractor,SiftExtractor
from pathlib import Path
import numpy as np
import cv2

if __name__ == '__main__':
    print('starting offline.py')
```

```
    print('Starting CBIR Extraction')
    bins=(8, 12, 3)
    CE = CbirExtractor(bins)
    #print('done makeing the feature extraction object')
    #print(sorted(Path("./static/img").glob("*.jpg")))
    for img_path in sorted(Path("./static/img").glob("*.jpg")):
        print(str(img_path),type(str(img_path)))  # e.g.,
./static/img/xxx.jpg
        feature = CE.extract(img=cv2.imread(str(img_path)))
        print(feature,type(feature))
        feature_path = Path("./static/CbirFeature") / (img_path.stem +
".npy")  # e.g., ./static/CbirFeature/xxx.npy
        np.save(feature_path, feature)
    print('Done CBIR Extraction')
    print('--------------------**********-----------------------')
    print('Now starting Sift Extraction')
    SE = SiftExtractor()
    #print(sorted(Path("./static/img").glob("*.jpg")))
    for img_path in sorted(Path("./static/img").glob("*.jpg")):
        print(str(img_path),type(str(img_path)))  # e.g.,
./static/img/xxx.jpg
        feature = SE.extract(img=cv2.imread(str(img_path), 0))
        feature_path = Path("./static/SiftFeature") / (img_path.stem +
".npy")  # e.g., ./static/SiftFeature/xxx.npy
        np.save(feature_path, feature)
    print('Done Making Sift Extraction')
```

3. server.py which is the code for running the flask api in the local host.

There are two routes in the server, one for sift and the other for CBIR

```
import numpy as np
from PIL import Image
from feature_extractor import SiftExtractor,CbirExtractor
from datetime import datetime
from flask import Flask, request, render_template
from pathlib import Path
import cv2

app = Flask(__name__)
```

```
# Read image features
bins=(8, 12, 3)
CE = CbirExtractor(bins)
SE=SiftExtractor()
CE_features = [] # all the CBIR features of the images
SE_features=[]
CE_img_paths = []
SE_img_paths=[]
#looping to get cbir features
for feature_path in Path("./static/CbirFeature").glob("*.npy"):
   CE_features.append(np.load(feature_path))
   CE_img_paths.append(Path("./static/img") / (feature_path.stem + ".jpg"))
#features = np.array(features)
# looping to get Sift features
for feature_path in Path("./static/SiftFeature").glob("*.npy"):
   SE_features.append(np.load(feature_path))
   SE_img_paths.append(Path("./static/img") / (feature_path.stem + ".jpg"))

# this is for CBIR retrieval
@app.route('/', methods=['GET', 'POST'])
def index():
   if request.method == 'POST':
       file = request.files['query_img']

       # Save query image
       img = Image.open(file.stream)  # PIL image

print('&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&**********************^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^')
       print(str(file.name),type(str(file.name)))
       uploaded_img_path = "static/uploaded/" +
datetime.now().isoformat().replace(":", ".") + "_" + file.filename
       img.save(uploaded_img_path)
       queryImage = cv2.cvtColor(np.array(img), cv2.COLOR_RGB2BGR)
       # Run search
       query = CE.extract(queryImage)
       #print('length of query',len(query))
       #print('length of all features',len(features))
       #print('length of one feature',len(features[1]))
       #print('type of features',type(features))
       #print('first_feature',type(features[0]),features[0])
       #print('query feature list',query)
```

```
        #print('------------------')
        #print('one feature list',features[1])
        #print('------------------')
        dists=[CE.chi2_distance(list(x),query) for x in CE_features ] #
getting the distances between the query image and the other images
        #dists = np.linalg.norm(features-query, axis=1)  # L2 distances to
features
        ids = np.argsort(dists)[:30]  # Top 30 results
        scores = [(dists[id], CE_img_paths[id]) for id in ids]

        return
render_template('index.html',query_path=uploaded_img_path,scores=scores)
    else:
        return render_template('index.html')
# this route for the Sift image retrieval
@app.route('/Sift', methods=['GET', 'POST'])
def indexSift():
    if request.method == 'POST':
        file = request.files['query_img']

        # Save query image
        img = Image.open(file.stream)  # PIL image

print('&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&**********************^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^')
        print(str(file.name),type(str(file.name)))
        uploaded_img_path = "static/uploaded/" +
datetime.now().isoformat().replace(":", ".") + "_" + file.filename
        img.save(uploaded_img_path)
        opencvImage = cv2.cvtColor(np.array(img), cv2.COLOR_RGB2BGR)
        img_cv2=cv2.cvtColor(opencvImage,cv2.COLOR_BGR2GRAY)

        # Run search
        query = SE.extract(img_cv2)
        dists=[SE.matching(x,query) for x in SE_features ]
        #dists = np.linalg.norm(features-query, axis=1)  # L2 distances to
features
        ids = np.argsort(dists)[:30]  # Top 30 results
        scores = [(dists[id], SE_img_paths[id]) for id in ids]

        return
render_template('indexSift.html',query_path=uploaded_img_path,scores=scores
```

```
)
   else:
       return render_template('indexSift.html')

if __name__=="__main__":
   app.run()
```

4. two html files for visualization

```
<!doctype html>
<html>
   <head> <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css
">



   </head>

   <body>

       <div class="container center_div">
         <div class="span4">
           <img class="center-block" width=500
src="{{url_for('static',filename = 'Images/logo3.png')}}">
         </div>

         </h1>

           <h2 class="text-center" >

             <b><i>CBIR</i> IMAGE SEARCH ENGINE</b></h2>
```

```
            <h2 class="text-center"><i>BY:</i>
            </h2>
            <h2 class="text-center"><small>Khalid Alabyad and Moaaz
Elsaeed.</small>
            </h2>
            <b>  <br><br><br><br> </b>
          <dev class="col-md-4 col-md-offset-4"><form method="POST"
enctype="multipart/form-data">
              <input type="file" name="query_img">
              <input type="submit">

          </form>

        </dev>
        <b>  <br><br><br> </b>
        <dev>
          {% if query_path %}
          <h3>YOUR QUERY:</h3>

          <img src="{{ query_path }}" width="300px">

          <h3>Results:</h3>
          {% endif %}
          {% for score in scores %}
          <figure style="float: left; margin-right: 20px; margin-bottom:
20px;">
              <img src="{{ score[1] }}" height="200px">
              <figcaption>{{ score[0] }}</figcaption>
          </figure>
          {% endfor %}
        </dev>
      </div>
   </body>
</html>
```

Discussion and Output:

- **Part 1 (SIFT class):**

It has 2 function: one for extracting features and the other for matching between the features of 2 images

-extracting the features starts by taking a grayscale image then apply `cv2.xfeatures2d.SIFT_create()` function to the image then outputs the feature matrix which is a (n,128)

-matching the features takes 2 feature matrices of the 2 images then apply flann.knnmatch to get the distances and returns them

We used the flann method as it is proved to be more optimized than other methods like euclidean distance, etc.

The output of the filtering is working perfectly and the output is in the results folder.

- **Part2(CBIR):**

This method is mainly getting the features from the 3d histogram of the images.

It has 3 functions : extraction function which also uses the histogram function, and the distance function which is chi2 distance function.

- **The extraction function goes in several stages**

  The first stage is converting the rgb scale to hsv scale which is better for the histogram as it mimic the human on this matter.

  Then we divide the image into 5 regions to get the histogram for every region: (1) the top-left corner, (2) the top-right corner, (3) the bottom-right corner,(4) the bottom-left corner, and (5) the center of the image as follows:

Then we get the histogram for every region and add them to a features list then returning them

- **The distance metric we use for 2 image features is chi2 distance which is supposed to be the best metric for the histogram**

$$X^2 = \frac{1}{2}\sum_{i=1}^{n}\frac{(x_i - y_i)^2}{(x_i + y_i)}$$

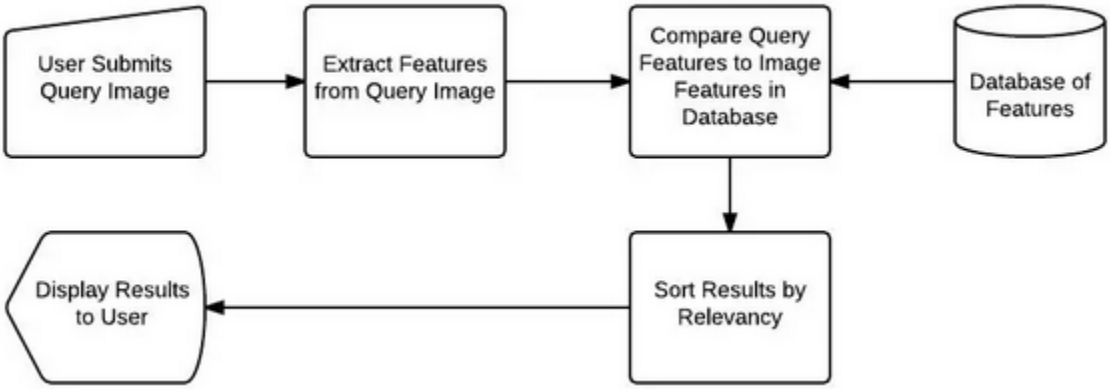
- **part3(extracting feature for the image database)**

  This process goes as

  We apply the feature extraction using the two methods over every image then stores the features in 2 seperate folders for every method.

  Part 4(Searching)

  Searching for an image goes by:



  Where the user uploads an image then we start by taking the image and extracting it's features based on the method specified then we load the features saved in the database then loop over the feature to calculate the distances and puts them in an array, then we present a specific number of images based on the scores from best to worst.

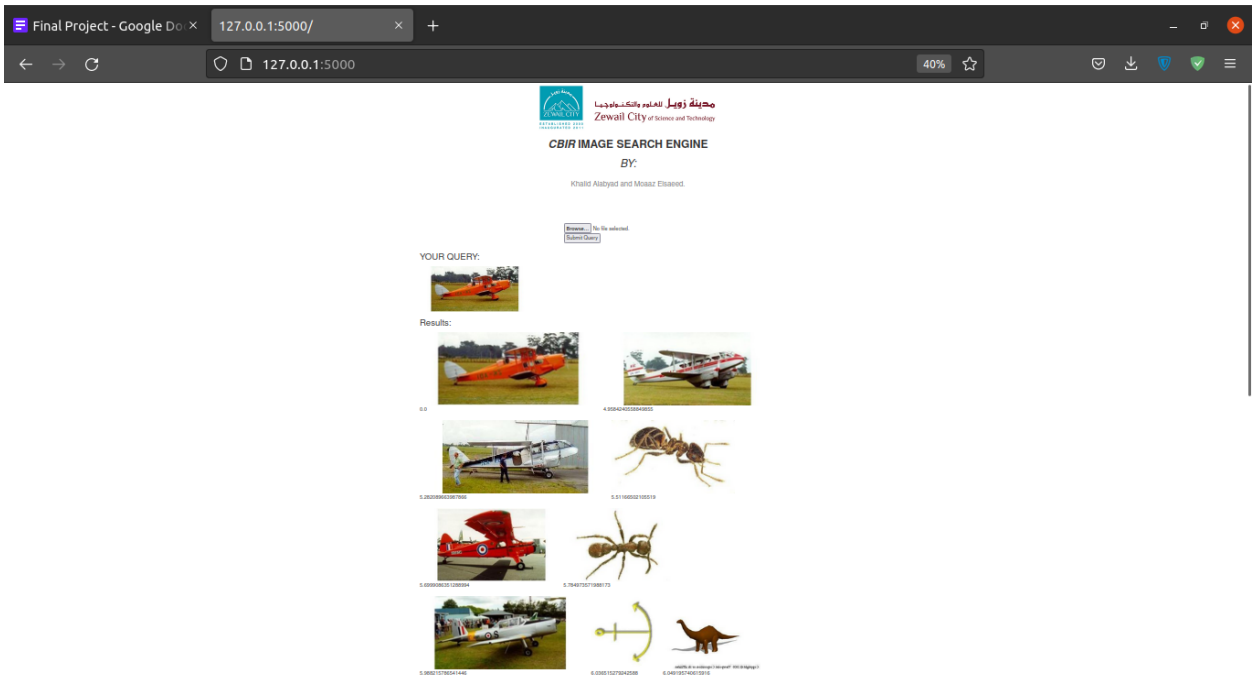
Results and discussion

To run the app we run server.py which make the server up in the local host.

We go to the local host and the output is an html page :

The result is the html page to the CBIR method.

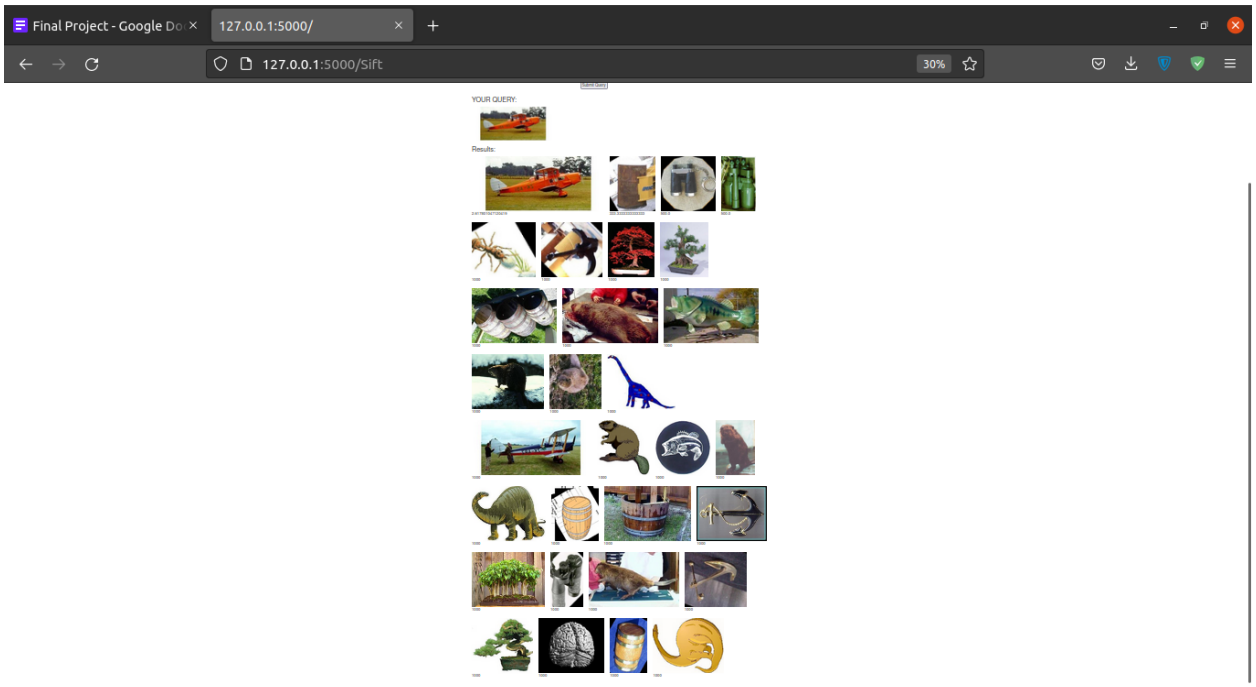
To upload an image we pressed on Browse and choose an image to search for then submit Query:



To try the SIFT Method we add '/Sift' to the URL.

The Output is

We searched with the same photo and we got:

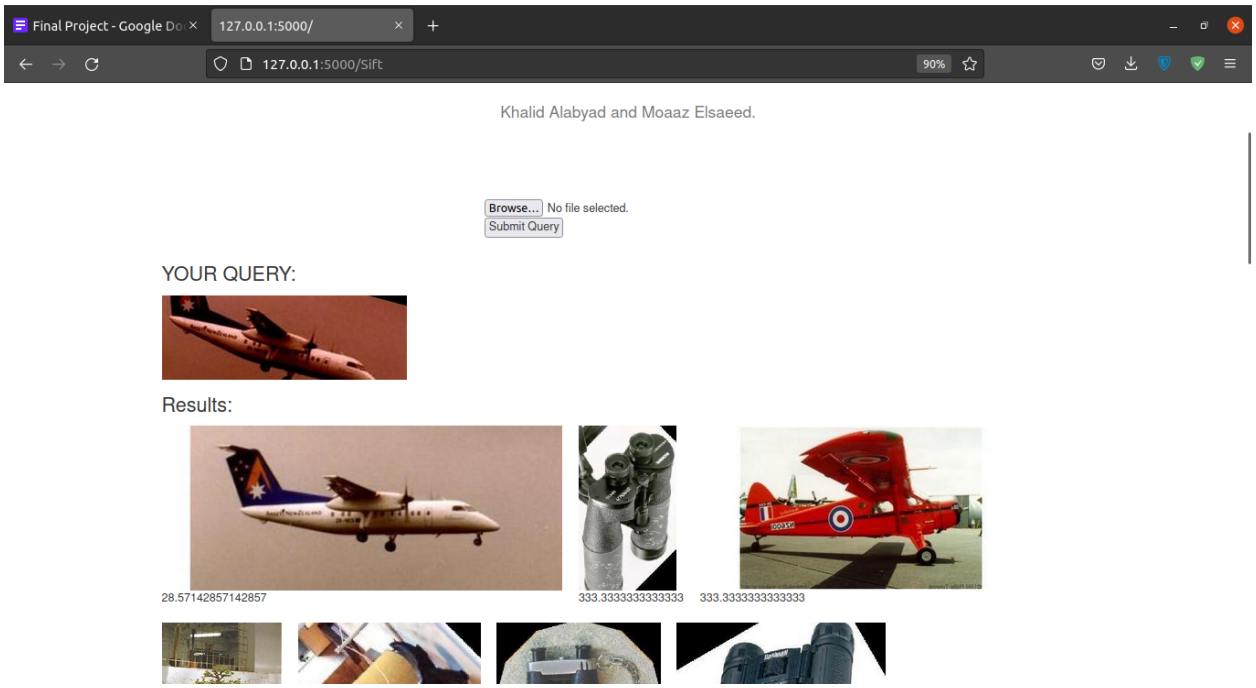


In the first look it would appear that CBIR is better.

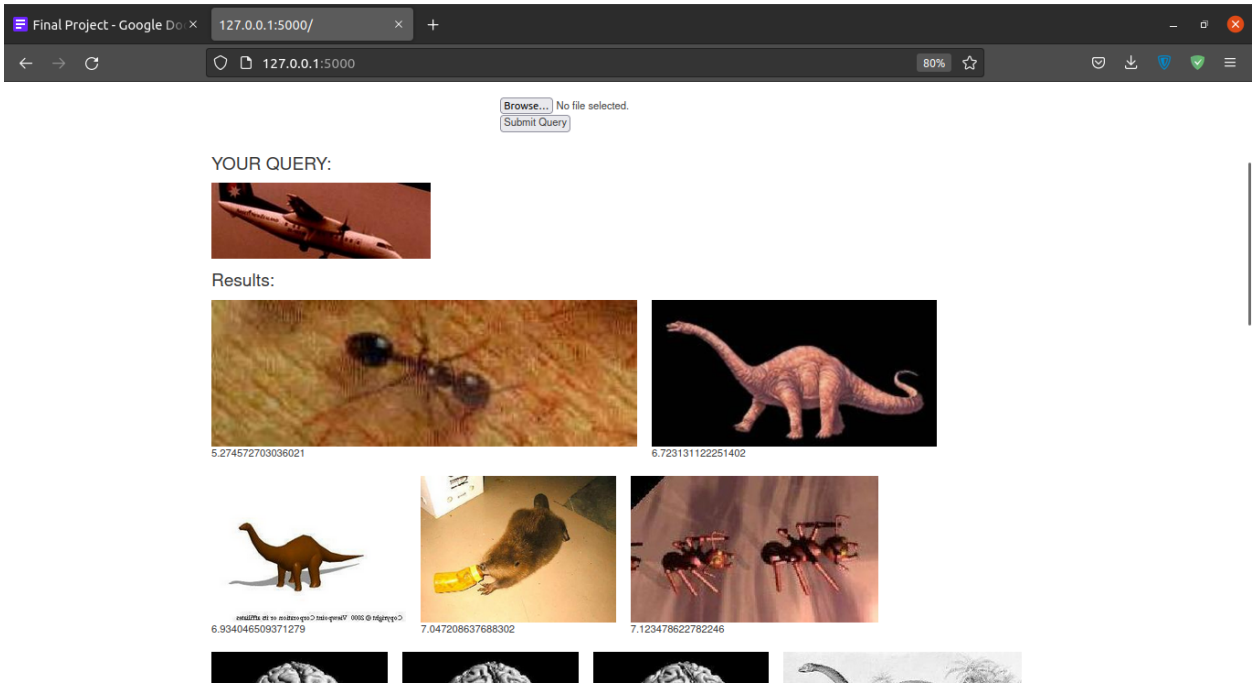
As both of them found the image itself in the database which is supposed to happen but CBIR got more photos related to the submitted photoo than the SIFT method.

But if we try to submit a photo that is edited we got this:

For SIFT



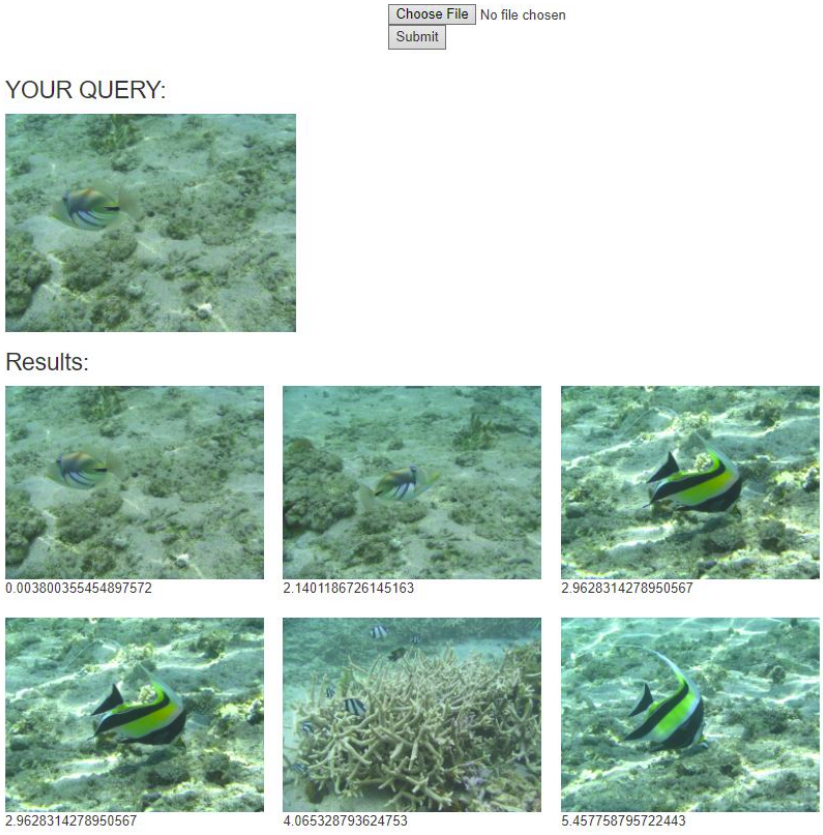
And for CBIR



And it appears SIFT was better as it found the same photo unedited but CBIR didn't. Which is also logical as sift is based on finding related features between the images. But CBIR just gets the texture of colors in the images.

So based on the type of Images every method can have it's pros and cons

For example when we tried to search on the Data by INRIA holiday it was a great success as the data is high quality images for tourism places



YOUR QUERY:

Results:

0.005340729938634046

4.674769068124666

5.074859388478447

5.748800822573835

6.688974453702768

7.220400002332673

8.035369487732845

8.41490161149586

8.506896841767325

YOUR QUERY:

Results:

0.0012369832595726417

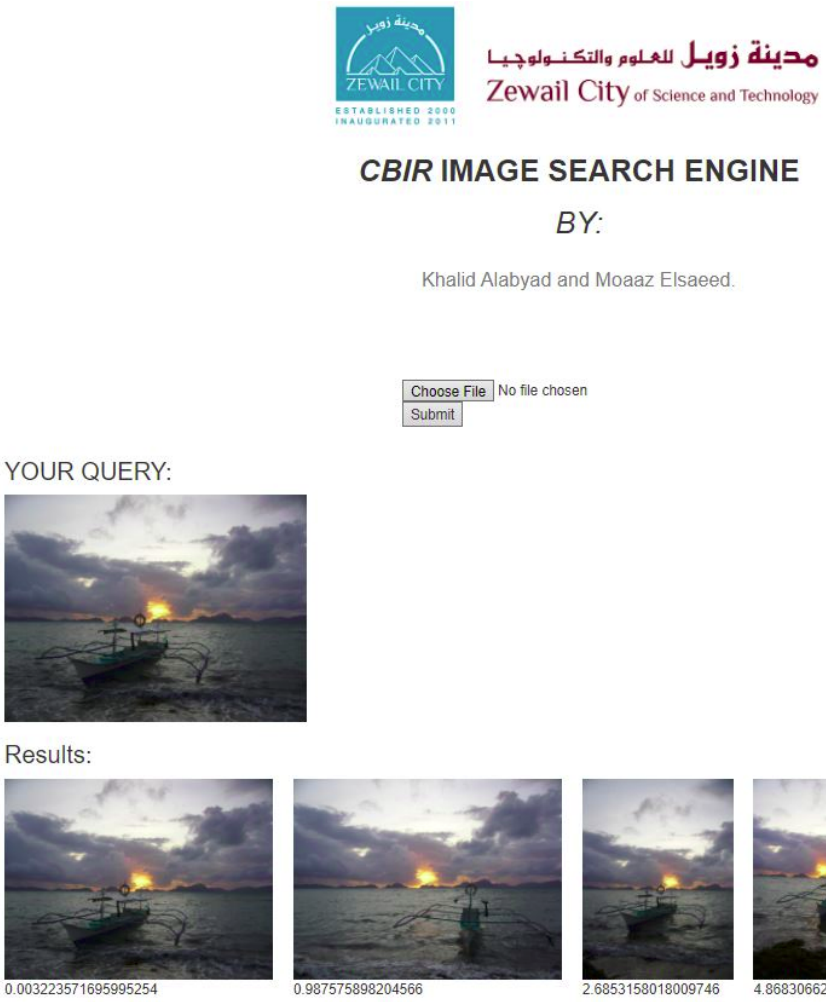
0.6407412735612619

2.115914584493101

3.170322575291637

3.675973758743697

3.8100936585986536



And as it seems the results is very good

But we couldn't try the Sift method as it generates a very large files for the features and the data is 2.6 GB and the features files is 21 Gb

And it appears that using CBIR is the best choice for its speed and storage

But we can make it even more accurate by adding both shape and texture with the color feature.

- **Other methods:**

  -We tried a pre-trained model by tensorflow called: VGG16 to get the features from the images and it got a much better result.

- [ ] **-We tried to make classification in order to make the search hybrid with feature extraction and classification. Which starts with classification first then applies feature scoring.**

  First we tried to implement classification model to search for an image in its class. the method used is Bag of words with Linear Support Vector as classifier. we tried different configurations for SVC and number of words. the most important step is to choose proper images descriptor. we tried as feature extractor: (SIFT, ORB and Cbir). All of them obtained poor accuracy with 9.4%, 4.5% and 2.5%. We believe that it was a bad decision to use these feature descriptors. We know that the best feature extractor (image descriptor) is HOG but we tried many configurations and it caused memory error.

References:

1-https://paperswithcode.com/dataset/caltech-101

2-https://lear.inrialpes.fr/~jegou/data.php

3-https://pyimagesearch.com/2014/12/01/complete-guide-building-image-search-engine-python-opencv/

4-https://vermaabhi23.github.io/publication/2017SIFT7.pdf