Machine Learning Engineer Nanodegree

Capstone Project: Image search Engine

Zhuo Chen

Jan 10th, 2017

Definition

Project Overview

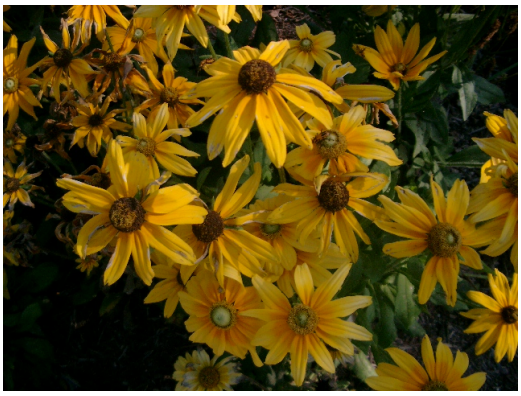
The objective of this project is to design and implement an image search engine and test the performance on UKbench dataset [1].

In UKbench dataset, each set of pictures contains four images of the same object. The goal of the system is for a given picture from the dataset, find out the at least another picture in the same set.

Problem Statement

The objective of this project is to design and implement an image search engine and test the performance on UKbench dataset [1]. For each object in the dataset, there are four images of it. The problem to be solved in this project is to develop a system that, for any given image in the dataset, find out the four images representing the same object. The image could be given in the form of the path of the image, or the index of it in the dataset.

For example, given the No.4964 image from the dataset:



It is desired that the system will return the No. 4964 to No. 4967 images, which representing the same group of objects.

The problem can be decomposed into the following steps::

1. Implement a feature extractor to extract the features of each picture
2. Apply the feature extractor to establish the feature database of the whole dataset
3. For each searching image, extract the features. Iterate through the database and find the four ‘closest’ images. There should be three other images, however, the image itself will also be searched and it will be counted as the ‘closest’ image. The distance metrics will be explained in later sections.
4. In order to tune and improve the performance of the system, it will first be tuned on a training set. The final performance will be evaluated on the testing set.

As we are looking for the same object and it looks similar in all the four image, the color distribution of the images should be similar. This indicates that we can compare the color distribution of the images to find out the same object. Thus, the solution to the first task could be extracting the histogram feature of the image. After all the histogram of the images in the

database are extracted, finding the image representing the same object is equivalent to finding the image with the closest histograms. The measurement of 'close' will be defined in later sections.

Metrics

The goal of this system is to find out all the other three images and the image itself. Thus, a very practical measurement of the performance could be accuracy, which is the number of the correct identified images versus total possible results.

There are many other possible options to evaluate the performance of a machine learning model. For example, precision and recall. However, for this model, the object is well defined, that is finding out all the 4 images and the most thing we care about is how many of the result is correct. Also, from the dataset website [1], the example is given in accuracy, which indicates the accuracy is a good candidate for evaluating the model.

Thus, the performance of the system will be evaluated by accuracy. There are four images for a same object in the dataset, a correct search would be, for a given image, all the other three images will be found.

Thus, for each search operation, the accuracy is:

$$Accuracy = \frac{number\ of\ correct\ results}{4}$$

If the feature extractor works properly, the image itself will have the closest distance. The best-expected performance will be 1 and the worst should be at least 0.25.

Analysis

Data Exploration

The dataset adopted for this project is the UKbench dataset [1]. The dataset can be downloaded from the following link:

http://vis.uky.edu/~stewe/ukbench/ [2]

A glance of the dataset is shown as Figure 1:

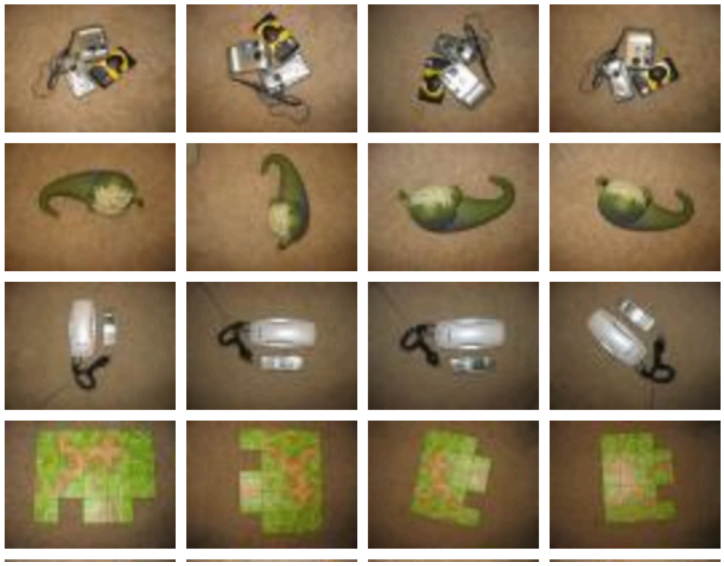


Figure 1 A few samples of the UKbench dataset

Some very important features of the dataset are:

1. Many of the images are focused on one object
2. For the same object, the images were taken in different directions with regards to the object.
3. For the same object, most of the images are taken with a similar background
4. For many of images of the same object, the light condition is similar.

The website mentioned that for each different subset of the dataset, the performance was different and plotted in the following image.

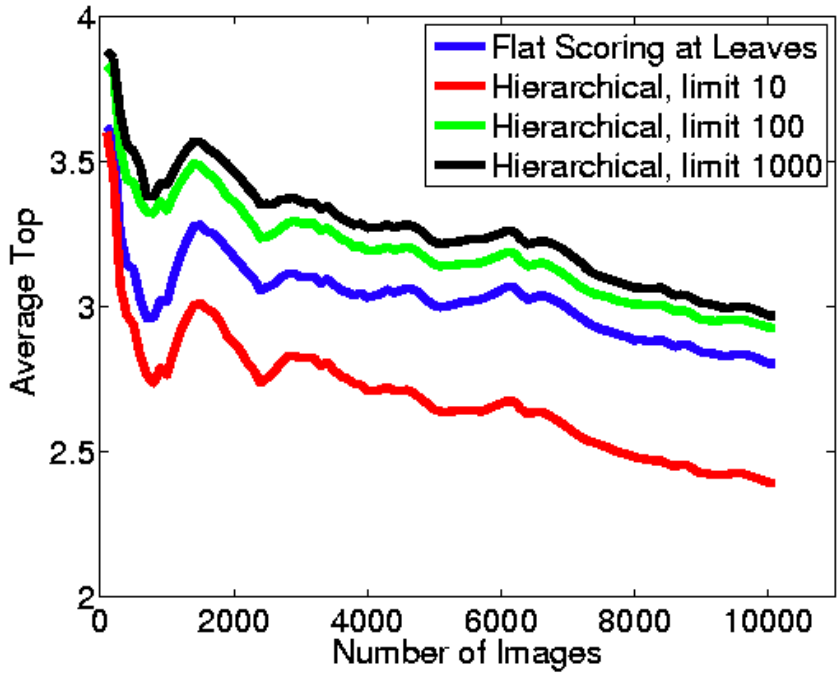


Figure 2 Plot of the performance versus subset [2]

Some information can be interpreted from this plot:

1. In general, the performance is decreasing alongside the number of images, which indicates the later subsets have higher difficulty.
2. The performances vary alongside the dataset, which indicates the samples are not distributed randomly in the dataset.

This project will target on the 'plateau' region of the plot above, which is 4000 to 5000, which could be suitable subsets to evaluate the performance and the difficulty of these subsets tend to be stable.

Also, it is necessary to split the training set and testing set randomly. The attributes of the samples data set are distributed un-randomly.

Exploratory Visualization

From the previous section, some features of the dataset was highlighted and these features could be suggestions that the color histogram could be a good candidate to expression the features of the image. A very common color representation is RGB, which the pixels have three

values each represents Red, Green, and Blue intensity. It a good format for storing and displaying images on the computer, however, there is another format, HSV, is more friendly for the way a human perceives color [3].

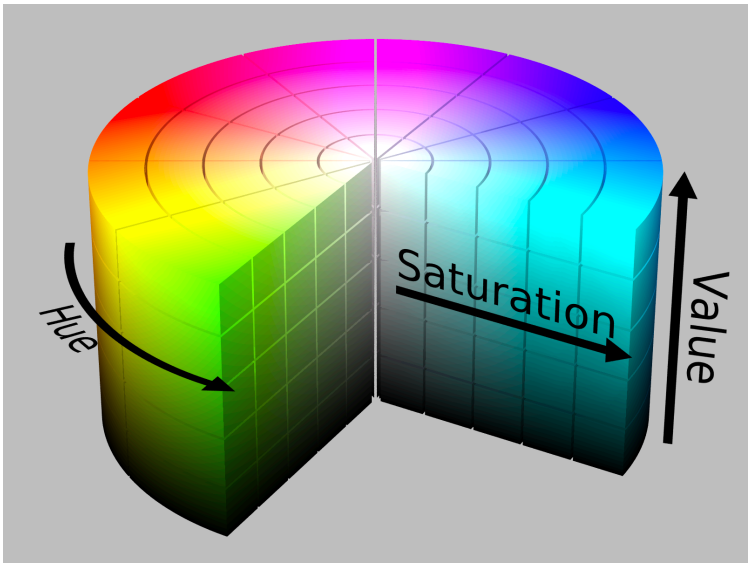


Figure 3 HSV cylinder [3]

Figure 3 is the plot of how colors are stored in HSV domain. Instead of R, G, B, the color is stored in terms of Hue, Value, and Saturation.

To make ann exploratory visualization, a sample image was selected from the dataset(No.50) as:

Figure 4 A sample image of the UKbench dataset

And the HSV histogram plot:

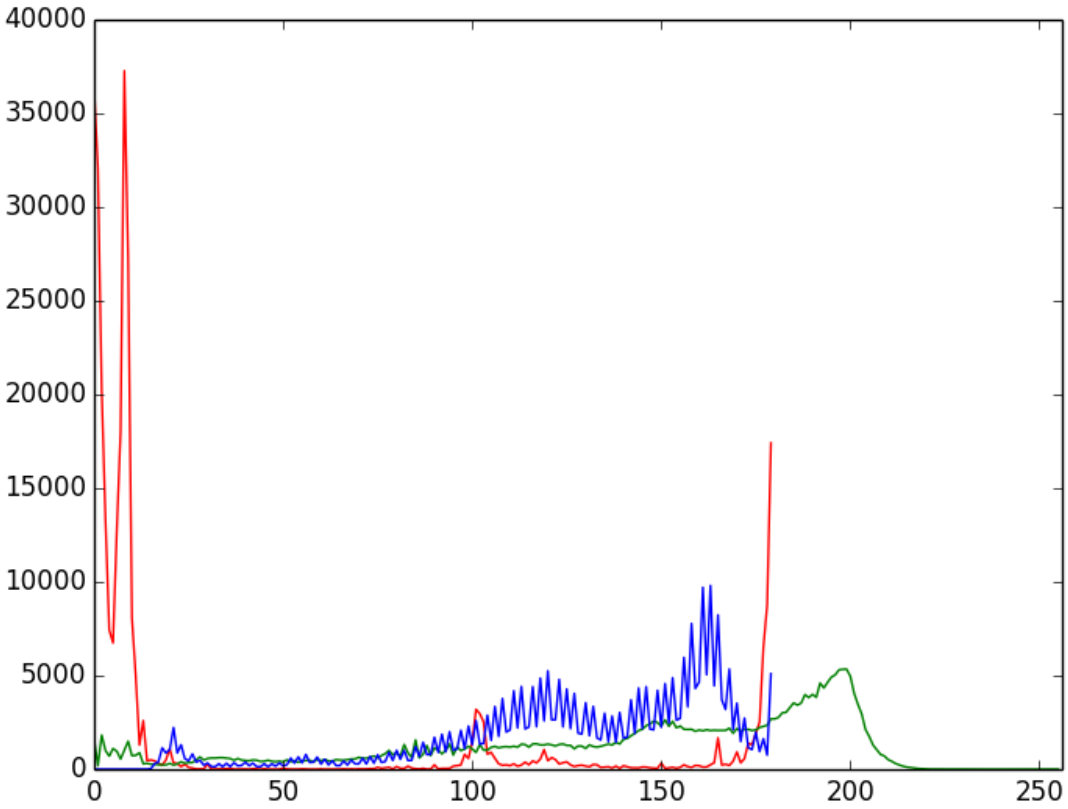


Figure 5 HSV histogram plot of No.50 image in the UKbench dataset

Algorithms and Techniques

Feature extraction

As the most important part of bridging computer vision and machine learning, the feature extraction process provided the chance of the features can be learned and described by the model.

In this project, the HSV histogram of the image will be extracted from the image. Rather than extracting the histogram of the full image, some features of the images in this dataset would help improve the performance.

First, the object is placed in the center of the image. By applying a rectangle mask to the image, we can extract the most of the object. Also, the background of the image is also similar, which can also be used as a reference.

The parameter for this process is the bin size of each channel. If the number of the bin size is too large, the model may tend to over-describing the current sample, leading to over-fitting. Also, if the number of the bin size is too small, the model may fail to describe the image [4].

This will be the main tuning parameter for the refinement process. The starting point of the parameter will be (2, 4, 4) for H, S, V as H has a range of (0, 180) and S, V have range of (0, 256).

Distance comparison

The distance metric used in this project is the chi-square [5]:

$$d(H_1, H_2) = \sum_I \frac{(H_1(I) - H_2(I))^2}{H_1(I)}$$

The chi-square [5] distance calculation is already implemented in openCV as:

cv2.compareHist(histA, histB, cv2.cv.CV_COMP_CHISQR)

this function will return the float as the distance.

Image search

The image searching process can be generalized as the following process:

1. Apply feature extractor on the dataset and store the output as an index file
2. Apply feature extractor on the searching image and obtain the feature vector.
3. Compare the object feature vector with the vectors in the index file, find the closest 4 samples

The searching process is very close to the classic K-NN method, however, it is not a typical clustering problem. In a classic K-NN problem, a sample will be clustered by finding the K nearest neighbors. Here the problem is finding the k (k=4) neighbors and assume they are in a same cluster.

Benchmark

When introducing the dataset, Figure 2 was used to demonstrate the uneven distribution of the samples in the data. It also provides a good reference for performance benchmarking. For flat score strategy, which is same as the one used in this project, the performance is around 3, which is 0.75 in our case (3/4). This could be set as the goal of this project. However, the feature extraction technic used is very naïve in Computer vision domain, it is expected that the performance would be much lower than given example. Thus, the bottom line set for this project is 0.5, where at least one other image can be found.

Methodology

General pipeline

The general workflow of the project can be summarized as following steps:

1. Implement a feature extractor to extract the features of each picture
2. Apply the feature extractor to establish the feature database of the whole dataset
3. For each searching image, extract the features. Iterate through the database and find the four ‘closest’ images. There should be three other images, however, the image itself will also be searched and it will be counted as the ‘closest’ image.

This is the pipeline given by Adrian [4] on his online tutorial.

Data Preprocessing

The dataset was divided into training(tuning) and testing set with code:

```
train, test = model_selection.train_test_split(range(250), random_state=1)
```

This will return the index of the sub-set selected to be the training and testing sample. For example, if 5 is in training set and the sample index starting from 4000, then picture No. 4020, 4021, 4022, 4023 will be training samples.

The random state is set to a fix number to ensure the train and set used for the tuning process is same everytime.

Implementation

Feature extraction

As the process are all carried out in HSV domain, we need to first convert the RGB image into a HSV image. The code for this purpose is:

```
image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
```

The right-side image is the original one, and the left-side is the image after HSV transformation.

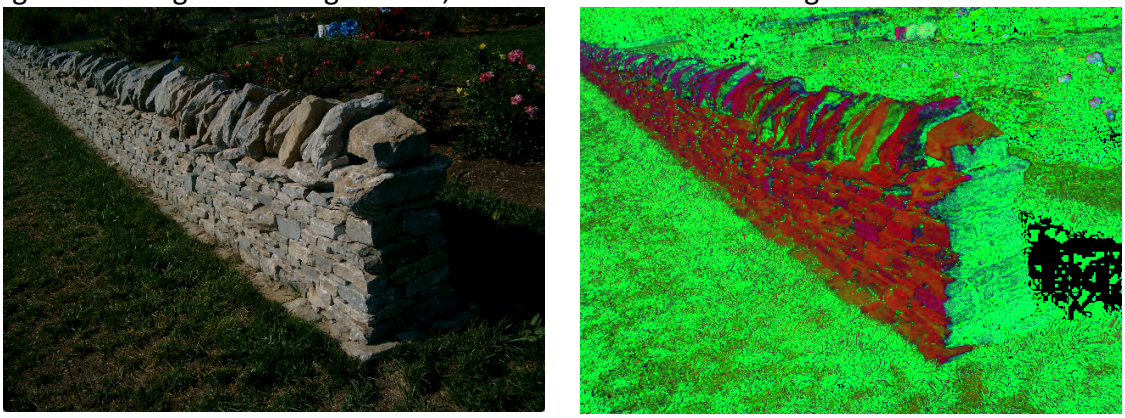


Figure 6 Original image and HSV transformed image.

Then we need to split the image into four segments:

```
segments = [(0, cX, 0, cY),  # upper left
            (cX, width, 0, cY),  # upper right
            (cX, width, cY, height),  # lower right
            (0, cX, cY, height)]  # lower left
```

After the segments are generated, iterate through the segments, extract the features and append them to the feature vector. An example of the upper right corner is given in Figure 7.

```
# add the four fragments' histogram to the feature vector
for (X0, X1, Y0, Y1) in segments:
    corner = np.zeros((height, width), dtype="uint8")
    cv2.rectangle(corner, (X0, Y0), (X1, Y1), 255, -1)
    corner = cv2.subtract(corner, rec)
    corner_hist = cv2.calcHist([image], [0, 1, 2], corner, bin_numbers, [0, 180, 0,
256, 0, 256])
    corner_hist = cv2.normalize(corner_hist).flatten()
    feature_vector.extend(corner_hist)
```

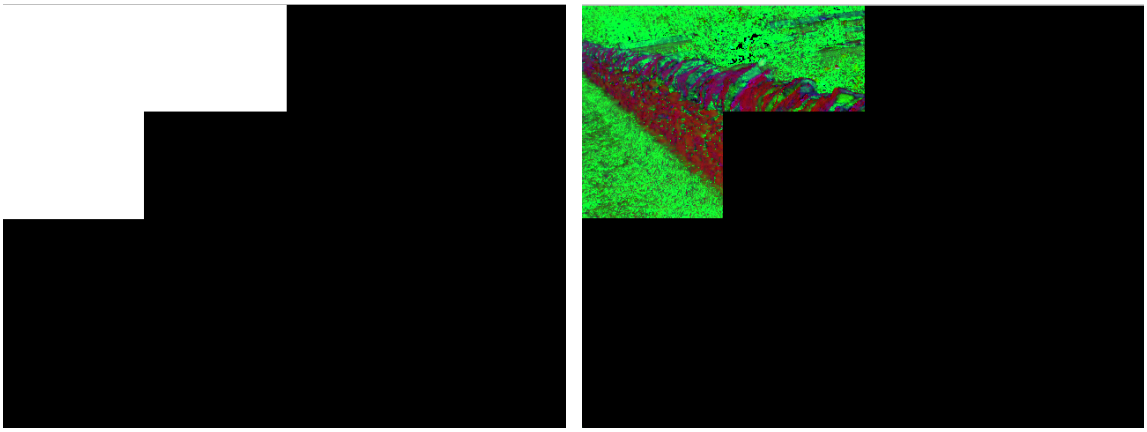


Figure 7 Corner mask and the effect of applying the mask on image

Finally, extract the center fragment's histogram and added to the feature vector.

```
center_hist = cv2.calcHist([image], [0, 1, 2], rec, bin_numbers, [0, 180, 0, 256, 0, 256])
center_hist = cv2.normalize(center_hist).flatten()
feature_vector.extend(center_hist)
```

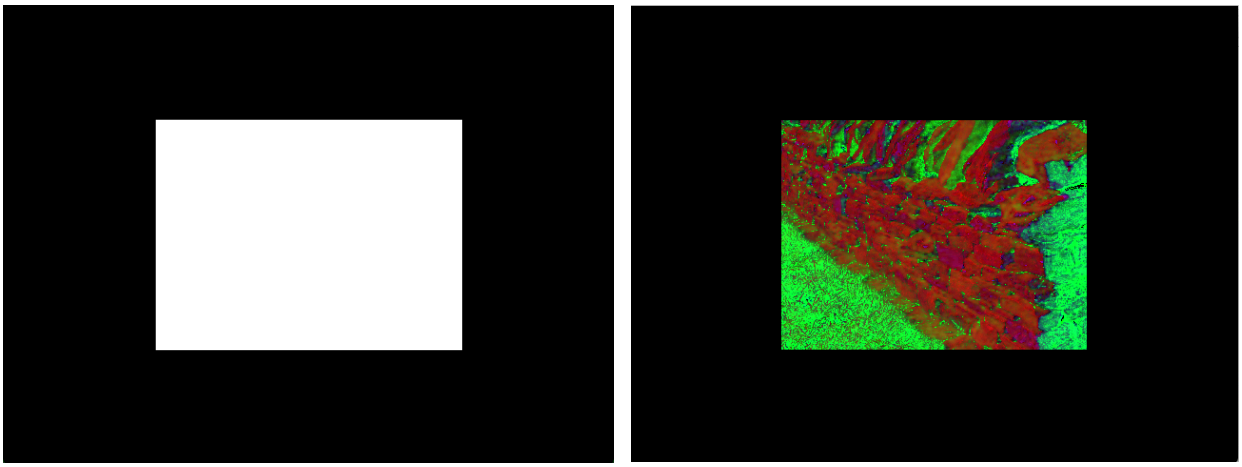


Figure 8 Center mask and the effect of applying the mask on image

Figure 7 and Figure 8 show the shapes of the masks and effect of applying the mask. All the histogram of the fragments will be calculated and stored as the feature vector of the image.

Distance comparison

OpenCV has already implemented chi-square method. However, the data type need to be converted to float32 to pass into this function.

```
histA = np.array(histA, dtype='float32')
histB = np.array(histB, dtype='float32')
d = cv2.compareHist(histA, histB, cv2.cv.CV_COMP_CHISQR)
```

Index the dataset

For the searching process, the feature of the target image will be compared with all the image. To speed up the process, rather than extract the feature of the image in the dataset every time, the features can be cached.

```
for (i, imagePath) in enumerate(images):
    filename = imagePath[imagePath.rfind("/") + 1:]
    image = cv2.imread(imagePath)

    features = descriptor.extract_feature(image, bin_numbers)
    features = map(str, features)
    output.write("{},{}\n".format(filename, ",".join(features)))
```

The code above will iterate through the dataset, extract the feature of each image, and write them into an index file.

Image searching

To perform the searching, first the features of the target image will be extracted:

```
image_to_search = cv2.imread(image_to_search)
features_to_search = extract_feature(image_to_search, bin_numbers)
```

Then this feature will be compared with the features in the index:

```
for row in reader:
    features = row[1:]
    features = map(float, features)
    d = chi2_distance(features, features_to_search)
    results[row[0]] = d

results = sorted([(v, k) for (k, v) in results.items()])
results = results[:number_of_results]
```

The comparison results will be stored in a dictionary. By sorting the dictionary values in ascending order, the desired images should be the first 4 elements in the dictionary. However, for the convenience of analysising the performance, the numbers of the returned result can be set manually, for example 6.

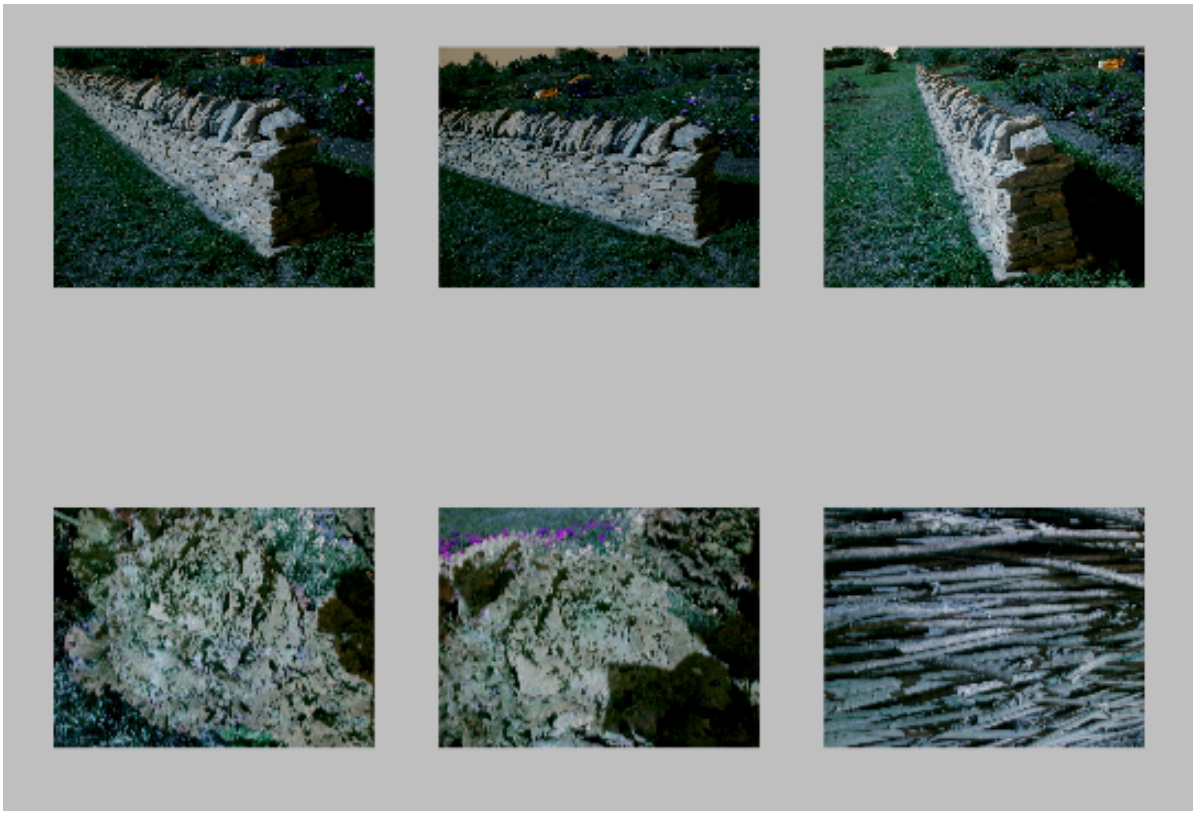


Figure 9 Searching result

To give a better demonstration, the first 6 results are selected rather 4 and some of the correct samples may appear the 5th or the 6th for a non-perfect search engine. Figure 9 is the result of the sample image. The first 3 results are correctly found out. However, the search engine failed to find the 4th image, it even did not appear till the 6th result.

Refinement

The parameter to be tuned for this model is the bin numbers of the histograms for Hue, saturation and value.
During the early stage of the implementation, a starting point was set to (2, 4, 4)
A table has been generated during the tuning process.

H	S	V	Accuracy
2	4	4	0.546
3	4	4	0.493
4	4	4	0.466
2	2	4	0.58
2	3	4	0.544
2	5	4	0.553
2	2	2	0.595
2	2	3	0.596

An assumption has been made that the effect of changing the H, S and V bin numbers are independent. For each feature, it was found out that the accuracy was decreasing with the bin number increasing and the resulted best parameter for the bin numbers are (2, 2, 3) and this gave an accuracy of 0.596.

Results

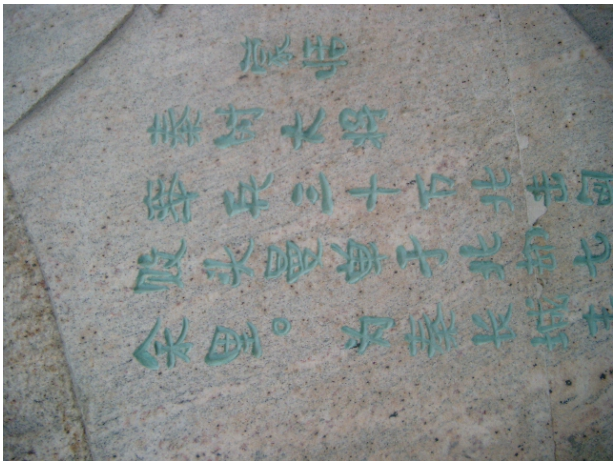
Model Evaluation and Validation

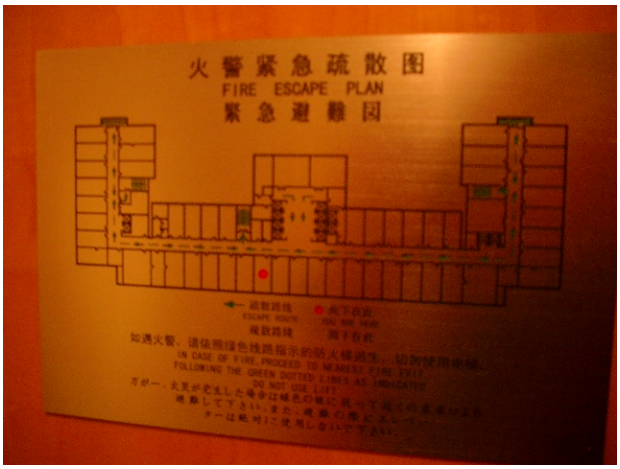
After tuning, the best parameter found for the HSV bin numbers is (2, 2, 3). Apply the model with this parameter to the testing set, the final accuracy is 0.558, which indicated that on average, at least one other image in the dataset of the same object can be found.

During the manually testing, which is randomly selected some samples to perform testing, it was found out that the model ‘s performance varies on different samples.

Below are some of the images scored 100% accuracy. Those images tend to be simply in color and dominated by a few colors with high contrast to each other. Also, many of them also follow the assumption that the object appeared in the center of the image.

Below are some examples of the images scores 0.25, which is the lowest possible score. Those pictures do not follow the assumption that the object is located in the center of the picture with a clear background. Those images are dominated by a few colors which have low contrast rate in H, S, V domain, which makes it hard for the model to distinguish them. This may explain the low performance on these pictures.





After tuning the HSV bin number parameter, the performance has increased from 0.546 to 0.596.

At this point, this model has meet the expected performance, which on average, for each searching image, at least one image can be found. However, this model only has a good performance on certain conditions, where the object is located in the center of the image and

has a clear background. Also, the ideal colors in the image should be simple and have high contrast in HSV domain.

Justification

The reference performance given in the benchmark is 0.75, better than the model in this project. However, the performance still meet the bottom line as expected. During the tuning process, the parameter obtained at last is (2, 2, 2). The bin number of each histogram is only 2, making each histogram can only provide binary information about the image. This also explained why this model only perform well on images will high contrast images.

The performance of generate did not meet the ultimate goal, but it met the bottom line. For each given sample, it can find at least one expected result. Also, this model has good performance on images with high contrast colors.

Conclusion

This project is carried out on the UKbench dataset, where for each object, there are 4 images representing that image in the dataset. A sample of the image is shown as below:

The object of the project is, for each given searching image, find out at least one image representing the same object.

For example, for image 4859, the searching result is shown below. 3 out 4 images have been found out.

Testing the model with the tuned parameter (2 ,2, 3), the final performance of the system achieved 0.54 in terms of accuracy, which indicates that on average, the object has been achieved.

Reflection and improvement

In summary, this project only went to a very simplified process of applying machine learning technics on computer vision area. Though this project only involved histogram distance and nearest neighbor method, it illustrated the basic workflow:

- Feature extraction with a proper feature extractor
- Feature comparison with a distance metrics
- Find the nearest neighbors with regards to the distances

However, it was found out that the histogram comparison is rather a naive technic for any slightly complicate task. The feature extraction process could be improved with some advanced descriptors for example, SIFT or SURF combined with some key point detectors.
When testing with one example, the searching speed is acceptable. However, during the tuning and testing progress, for each sample the program need to iterate through the whole dataset, it took around minutes to finish the iteration. The size of the dataset is only 1000. For practical use, the size of the dataset could achieve millions or even billions. Some more searching technics, for example reverse index, could be helpful to improve the searching efficiency.

Bibliography

[1] Henrik Stewénius, David Nistér, " Scalable recognition with a vocabulary tree," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR), ,volume 2,* pp. 2161-2168, Jun 2006.

[2] H. Stewénius, "UKbench dataset," [Online]. Available: http://vis.uky.edu/~stewe/ukbench/.

[3] Wiki, "HSL and HSV," [Online]. Available: https://en.wikipedia.org/wiki/HSL_and_HSV. [Accessed 15 Jan 2017].

[4] A. Rosebrock, "The complete guide to building an image search engine with Python and OpenCV," 1 Dec 2014. [Online]. Available: http://www.pyimagesearch.com/2014/12/01/complete-guide-building-image-search-engine-python-opencv/. [Accessed 11 Jan 2017].

[5] openCV, "Histograms," [Online]. Available: http://docs.opencv.org/2.4/modules/imgproc/doc/histograms.html.