CSE 515: Offline Image based Search Engine (Phase-1)

Abhishek Jindal (ajinda10)

Group Members:

Mohd Zaid

Dhruv Agja

Swaminathan Balachandran

Kiriti Ganga

Abstract:

The project incorporates Phase -1 of course project under CSE 515 Multimedia and Web Databases. This phase of the project not only focuses on similar image retrieval for greyscale images but also focuses on different models/similarity measures to retrieve these images. This phase discusses the similar image retrieval based on models such as Color Moment, Extended Local Binary Patterns (ELBP) & Histogram Over Gradients (HOG) and similarity measures such as Euclidian Distance, Chi Square Distribution measure & Earth-Mover's distance.

Keywords:

Olivetti Face Images, Color Moment, Extended Local Binary Patterns (ELBP), Histogram Over Gradients (HOG), Chi Square Distribution, Earth-Mover's distance

Table of Contents

Introduction:

The project incorporates Phase -1 of course project under CSE 515 Multimedia and Web Databases. I have named this project as Offline Image Based Search Engine. In this project we are trying to search k similar images based on models such as Color Moment, Extended Local Binary Patterns (ELBP) & Histogram Over Gradients (HOG) and similarity measure such as Euclidian Distance, Chi Square Distribution measure & Earth-Mover's distance. This is offline based as we already have the image corpus with us. The images are parsed and their image pixels are computed. Further the image descriptors and their similarity measures are computed as explained in the Solution/Implementation phase.

1. Terminology: The similarity measure is often coined with term "distance measure". It simply means how far (hence, the term distance) are the images from each other. The more the distance, the less similar are the images to each other.
2. Goal Description: The goal of the project is to get k similar images based on an input image. The goal not only incorporates the orientations of same person but also different versions/orientations of different people.

3. Assumptions:
    a. No RGB image will be provided in the input.
    b. The running environment would contain the libraries to support the execution of the code.
    c. HOG library is using unsigned gradients.

Solution/Implementation:

The k similar image retrieval is a non-trivial task which demands in depth knowledge of the different models to extract features and compare these feature descriptors using similarity measures. In this phase of the project, we are only working on greyscale images (for simplicity) and used the following models & similarity measures to compute the distance between 2 images.

1. Colour Moment Introduction:

Color moment by definition measures the color distribution (range of pixel values) within the image. This phase only deals with greyscale images and hence the range of pixels is 0-255. We are using the following three-color moments in this phase.

a. Mean Colour Moment:

This color moment is simply the average of all the pixels in the image. This gives us the idea of average color of the image. So, if two images have similar average color, there is similarity in the images.
This is defined by the following formula:

$$E_i = \sum_{j=1}^{N} \frac{1}{N} p_{ij}$$

b. Standard Deviation:

This color moment gives us the idea of variation of pixels with respect to mean i.e., whether the pixel distribution is close to mean or away from it. So, if two images have same standard deviation, that means the color distribution is same for both of them with respect to their mean. It doesn't mean that the pixel values are some but only talks about the dispersion of data. Hence, if two images have large difference in their standard deviation, that means the spread of pixels is not same for both the images. This is given by the formula.

$$\sigma_i = \sqrt{(\frac{1}{N}\sum_{j=1}^{N}(p_{ij} - E_i)^2)}$$

c. Skewness:

This color moment gives an idea about the concentration of data with respect to mean. Skewness can be positive, negative or zero. Negative skewness means the data is concentrated more towards the right of the mean. Positive skewness indicates the vice-versa. Zero skewness indicates the concentration of data on both sides of mean is same. This color moment also helps in computing the similarity between two images based on concentration of data with respect to mean. This is given by the formula.

$$\tilde{\mu}_3 = \frac{\sum_i^N \left(X_i - \bar{X}\right)^3}{(N-1) * \sigma^3}$$

2. Colour Moment Implementation:

In order to calculate the color moment feature descriptor, I have implemented three function to calculate mean, standard deviation and skewness of the image pixel values. The image is first broken down into 64 blocks of 8x8 each and localized mean, standard deviation and skewness are calculated. I am concatenating the localized mean, standard deviation and skewness in a vector of size 3. Hence, this results in a 3d feature descriptor of 8x8x3.

3. Colour Moment Similarity Measure:

I am using Euclidean distance for computing the distance between 2 color moment feature descriptors. I am first calculating the distance between mean vectors followed by standard deviation vectors followed by skewness vectors. The mean vector is defined by 64 localized means of each block. The same logic is followed for standard deviation and skewness. The formula for Euclidean distance is:

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

In the end, the total distance is weighted average of mean, standard deviation and skewness. For example: let's say d1 is distance between mean vectors, d2 is the distance between standard deviation vectors and d3 is the distance between skewness vectors, the total distance is:

$$D = m1 * d1 + m2 * d2 + m3 * d$$

where m1, m2, m3 are the weights given to each color moment.

I have taken the following values of weights for the color moment:

$$m1 = 0.3, \quad m2 = 0.35, \quad m3 = 0.35$$

I have given higher weightage to m2 and m3 because dispersion and concentration of data (w.r.t to mean) are more important than the mean of the distribution. This is used to retrieve k similar images in Task-3.

4. Extended Local Binary Patterns (ELBP) Introduction:

LBP takes care of local representation of the texture. This representation is computed by comparing its pixel with its surroundings. Here we take every pixel and threshold it against its n neighbors in a radius of size r. I have used the following values of n and r for my computation.

$$N = 24, \quad r = 8$$

5. ELBP Implementation:

I have used the LBP library present in SkImage to compute the corresponding feature descriptor. I have used the uniform distribution for computing my feature descriptor as the uniform distribution is rotation invariant. This simply means that we would be traversing the bits in circular fashion. After that the histogram is created out of this ELBP vector.

6. ELBP Similarity Distance Measure:

I am using chi square distance measure to calculate the distance between two ELBP feature histograms. This takes two histogram containing n+2 bins (because of uniformity) and measures how these histograms would be related to each other. We are using chi square distance as it clearly tells us the statistically significant relationship between 2 vectors. This is used to retrieve k similar images in Task-3.

The formula is given by:

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

7. Histogram Over Gradients (HOG):

HOG is a very interesting and important model in computing the similarity between images. HOG incorporates the gradient descent of the image pixels. This model is interesting not only because it takes care of change of pixels over the image, but it also

helps in compressing the data into a histogram of angular bins which makes the comparison easy. The further explanation is detailed in the implementation section.

8. HOG Implementation:

I have used the HOG library to compute the feature descriptor for the image. The library takes the following parameters for the computation of the feature descriptor.

$* number\ of\ orientation\ bins\ =\ 9$

$*\ cell\ size\ =\ 8$

$*\ block\ size\ =\ 2$

$*\ oriented\ gradients\ (yes/no)\ =\ no, and\ 0\ otherwise.$

$*\ L2-norm\ clipping\ threshold\ =\ 0.2$

The algorithm divides the image into cells of size 8x8 (resulting in 64 cells), further dividing each cell into a block of size 2x2 to calculate the gradient descent of each block with 50% overlap. The 50% overlap results in 49 blocks (7x7) rather than 16 blocks of (4x4). In order to calculate the gradient of each block, the library is using the following filters in x and y direction:

Filter in X direction: $[-1, 0, 1]$

Filter in Y direction: [-1

0

1]

After computing the direction and magnitude of the gradient descent of each block, the gradient vectors are stored in a 9-bin histogram. The 9 bin histogram results in corresponding angle range of 20 degree each (0-20, 20-40…160-180). This is the beauty of HOG algorithm, compressing the 64 image pixels information (8x8) into a histogram of 9 angular bins. In the end, the algorithm combines the HOG descriptor for all the cells.

The dimensionality of HOG feature vector is 7x7x2x2x9 where 7x7 is the number of blocks (with 50% overlap), 2x2 is the block size within a cell and 9 is the number of histogram angular bins.

9. HOG Similarity measure:

I am using my own implementation of earth movers' distance in computing the similarity between two HOG image descriptors. I am using my own implementation because I didn't find any library which could implement it for me as per my understanding of the algorithm. In this measure, I am computing the Euclidian distance between each bin of HOG descriptor to the other corresponding bin of the HOG descriptor and concatenating these distances to calculate earth mover's distance. In my understanding, this tells me the distance between each corresponding bin (pile) and how much I will have to move the distance to match the other corresponding bin. This is used to retrieve k similar images in Task-3.

10. Combination of all the models:

In this measure I am normalizing all the feature descriptors (models Colour moment, ELBP and HOG) as they lie in different ranges. I have used the following formula to normalize the values.

$$X(Normalized) = \frac{X - \text{Min}(Xi \dots Xn)}{Max(Xi..Xn) - Min(Xi \dots Xn)}$$

After normalizing, I am combining these three feature descriptors with weighted average of all the distances. The formula is given by:

$$d = m1 * c1 + m2 * e1 + m3h1$$

Where c1, e1 and h1 are the Colour moment, ELBP and HOG image descriptors. I am using the following weights for this computation.

$$m1: 0.3, \qquad m2: 0.35, \qquad m3: 0.35$$

I am giving more weights to the ELBP and HOG feature descriptors as these descriptors focus on the texture and gradient descents whereas colour moments focus more on structure of the image. This is used to retrieve k similar images in Task 4.

Interface Specifications:

The interface of the tasks is console based. All the input and outputs are presented on the console. Please refer to the following for further details:

Task 0:

- **Input:** No Input Required
- **Output:** Retrieves all the Olivetti faces dataset and saves the images pixels in the DB.

Task 1:

- **Input:** Image Id, Model Name
- **Output:** Prints the feature descriptor of the given image based on model name.

Task 2:

- **Input:** Folder Path containing images
- **Output:** Computes all the feature descriptors (Colour Moment, ELBP & HOG) and saves it to the DB.

Task 3:

- **Input:** Image Id, Model Name, k (number of similar images)
- **Output:** Prints k most similar images label along with their distance score.

Task 4:

- **Input:** Image Id, k (number of similar images)
- **Output:** Combines all the models and prints k most similar images label along with their distance score.

Installation Steps:

The following packages are required to run this project

- Python 3
- Pip 3
- Sklearn Library
- Scikit-image
- Numpy
- PIL
- PyMongo Library (Mongo Client)
- Mongo DB

Getting Started

- Install Python & Pip
    - Please refer to https://www.python.org/downloads/ for downloading Python 3.x
- Install Libraries Sklearn, Scikit-image, Pymongo
    - Use the following commands to install these libraries
        - pip3 install sklearn
        - pip3 install scikit-image
        - pip3 install pymongo
        -
- Install Mongo DB
    - Please refer to https://www.mongodb.com/try/download/community for installing Mongo DB on your machine
- Execution
    - The project is divided into the following 5 tasks.
    - Task 0:
        - This task retrieves Olivetti Faces from Sklearn dataset and stores them into the Mongo Database.
    - Run Command
        - python main_task_0.py
    - Task 1
        - This task takes image id as the input and retrieves the corresponding image pixels from MongoDB (which were saved in task-0). The task also takes one of the model names (Color Moment, ELBP, HOG) as the input and outputs the corresponding feature descriptor.

- Dependencies
  - Successful Execution of Task-0

- Run Command
  - python main_task_1.py

- Task 2
  - This task takes folder path as the input and loads all the images inside the folder. It further computes the color moment, ELBP and HOG feature descriptor for each image and stores it into the MongoDB.
  - Run Command
    - python main_task_2.py
- Task 3 & 4
  - This task takes folder base path, image name and one of the model names (Color Moment, ELBP, HOG, All) as the input. It queries the MongoDB for the corresponding folder base path and loads all the image feature descriptors for that base path. It further computes the similarity measure based on the input model and outputs K similar images based on the similarity measure.

  - **Please note that task 3 and 4 are combined in one file. If the user inputs 'All' as the model's name, the code will combine all the models and outputs k similar images.**
  - Dependencies
    - Successful Execution of Task-2
  - Run Command
    - python main_task_3_4.py

Execution Results:

```
Set 1 Output:
Input: set1 image-0.png cm8x8 4
Output: K Similar Images [('image-7.png', 2.5562719086368633), ('image-
6.png', 3.153055538039505), ('image-8.png', 3.2388438850759584), ('image-
2.png', 3.3937044239413434)]

Input: set1 image-0.png elbp 4
Output: K Similar Images [('image-3.png', 0.011291337703589773), ('image-
2.png', 0.01172178061555229), ('image-9.png', 0.011848723128501715),
('image-7.png', 0.014249611602691135)]

Input: set1 image-0.png hog 4
Output: K Similar Images [('image-7.png', 10.558794097471921), ('image-
2.png', 10.814162284599155), ('image-6.png', 11.21902250652846), ('image-
5.png', 13.155356076084265)]

Input: set1 image-0.png all 4
Output: K Similar Images [('image-7.png', 0.05658273588933899), ('image-
2.png', 0.11646836389806983), ('image-6.png', 0.21251724763692098),
('image-3.png', 0.47899142551394125)]
```

```
Set 2 Output:
Input: set2 image-0.png cm8x8 4
Output: K Similar Images [('image-2.png', 3.3937044239413434), ('image-
10.png', 4.128961362386017), ('image-11.png', 4.179552651027335), ('image-
13.png', 4.189593413624311)]

Input: set2 image-0.png elbp 4
Output: K Similar Images [('image-3.png', 0.011291337703589773), ('image-
2.png', 0.01172178061555229), ('image-4.png', 0.017435485500410436),
('image-1.png', 0.022646529400950712)]

Input: set2 image-0.png hog 4
Output: K Similar Images [('image-2.png', 10.814162284599155), ('image-
1.png', 13.864172985899756), ('image-10.png', 13.890870290641494), ('image-
12.png', 14.122026084913475)]

Input: set2 image-0.png all 4
Output: K Similar Images [('image-2.png', 0.0030307409133059 49), ('image-
3.png', 0.4330718832210342), ('image-4.png', 0.5485372399375548), ('image-
1.png', 0.6390057340029911)]
```

```
Set 3 Output:
Input: set3 image-0.png cm8x8 4
Output: K Similar Images [('image-60.png', 3.4151822655123487), ('image-
110.png', 3.422101958576855), ('image-100.png', 3.4813199138561366),
('image-90.png', 3.7895243239683065)]

Input: set3 image-0.png elbp 4
Output: K Similar Images [('image-110.png', 0.01125932009225235), ('image-
100.png', 0.01322706650658784), ('image-40.png', 0.013553349912927675),
('image-80.png', 0.016406486799254986)]

Input: set3 image-0.png hog 4
Output: K Similar Images [('image-130.png', 13.531889791447748), ('image-
100.png', 13.57271195619915), ('image-110.png', 13.668102654815652),
('image-10.png', 13.890870290641494)]

Input: set3 image-0.png all 4
Output: K Similar Images [('image-110.png', 0.02772057343971898), ('image-
100.png', 0.04046255323594948), ('image-130.png', 0.33670544937901586),
('image-60.png', 0.4131037366570171)]
```

Conclusions:

1. Colour Moment vector focuses on the structure of the image. For example, if we give a front facing image as an input, it will try to find k similar images of front face first. This is evident from Set 2 Image-0 results for colour moment vector.
2. ELBP focuses on the texture representation and hence is rotation invariant. Hence, no matter the orientation of the face, it will try to find the same texture images. This is proven by set2 results for image-0.png
3. HOG focuses on gradient descent of the pixels. It helps in recognizing the images with same intensity difference between the images. This is proven by almost every input set image.

References:

Wikimedia Foundation. (2021, June 14). *Color moments*. Wikipedia. Retrieved September 16, 2021, from https://en.wikipedia.org/wiki/Color_moments.

Dayala, R., & Dayala, R. (2020, July 23). *10.7 local binary patterns*. Computer Vision. Retrieved September 16, 2021, from https://cvexplained.wordpress.com/2020/07/22/10-7-local-binary-patterns/.

Feature descriptor: Hog descriptor tutorial. Analytics Vidhya. (2020, May 10). Retrieved September 16, 2021, from https://www.analyticsvidhya.com/blog/2019/09/feature-engineering-images-introduction-hog-feature-descriptor/.

Histogram of oriented gradients¶. Histogram of Oriented Gradients - skimage v0.19.0.dev0 docs. (n.d.). Retrieved September 16, 2021, from https://scikit-image.org/docs/dev/auto_examples/features_detection/plot_hog.html.

Pietikäinen, M. (n.d.). *Local binary patterns*. Scholarpedia. Retrieved September 16, 2021, from http://www.scholarpedia.org/article/Local_Binary_Patterns.

Waheed, A. (2020, October 30). *How to apply hog feature extraction in python*. Python Code. Retrieved September 16, 2021, from https://www.thepythoncode.com/article/hog-feature-extraction-in-python.