MWDB_PROJECT_1

Group: Sritej Reddy Nallamaddi
Pritam De
Aaron Steele
Ayush Anand
Shubham Verma
Sairaj Menon

Abstract :

In this project, we will perform 4 tasks individually. Before moving on to the four tasks we were asked to familiarize ourselves with olivetti faces dataset and then store the images from it in a file structure. The first 2 tasks revolve around extracting the Image features using models like *Color Moments, Extended local binary pattern, Histograms of Oriented Gradients* and then printing or saving them in a file. In the task-3, we need to retrieve the top 'K' images based on the similarity metric chosen for the given model. Whereas in the task-4, we need to combine the similarity distances from all the 3 models and then retrieve the top 'K' images in a given folder.

KeyWords :

- Olivetti faces dataset
- Color Moment
- Mean
- Standard Deviation
- Skewness
- Extended Local Binary Pattern(ELBP)
- Histograms Of Oriented Gradients(HOG)
- Features
- Image ID
- Similarity distance
- Weightage mean
- Matching score

Introduction:

This projects goal was to help us understand image features extraction using different models and metrics to generate the required numer of top ‘k’ matching images. We were required to get familiar with the olivetti faces dataset as our primary task. The task-1 mainly dealt with extracting of features from a given image based on Image Id and the given model and later print the extracted features as output. For instance, if an image Id, and a model like color moment is given as input, we extract the feature descriptors like mean, standard deviation, and skewness for each 8*8 block in the 64*64 image. The task-2 deals with feature descriptors extraction for all the images in a given folder and storing them in a json format file. Task-3 includes image Id, model and a folder of images as input for generating an output of the top ‘k’ images in the folder, depending on their matching score with the given image. In Task-4, we do the above steps, but instead of specifying a particular model, we have to generate the matching score for all models and then generate an overall matching score based on the weightage mean of scores obtained from above models.

Task-0

Goal: Task 0 is simply about downloading the images from Olivetti dataset and storing them in which ever way we choose.

Approach: I chose to store them in a folder, since its easy to access the images from folder when we execute the code.
And we could even store the images in array format to avoid accessing the folder repetitively for performing tasks over images data.

Dataset: There are ten different images of each of 40 distinct subjects which makes it a total of 400 images in the dataset. All the images were taken against a

dark homogeneous background with the subjects in an upright, frontal position (with tolerance for some side movement).

The images are quantized to 256 grey levels and stored as unsigned 8-bit integers; the loader will convert these to floating point values on the interval [0, 1], which are easier to work with for many algorithms.

Above attached are images of some of distinct subjects.

Task-1

Goal: It is to develop a function which takes Image ID and Model as parameter and generates respective Feature descriptors as output.

Approach: We have three models to choose from for extracting the features from an image.

1. Color Moment: It is a model which depends on the colour distribution of Image. It works really well if we want to compare two images based on color. Since our images are in grayscale we will have 3 color moments.

   Each of these 3 moments will be calculated for 8*8 blocks

    1. *Mean*: Calculating the average for pixel values in the 8*8 block(sub_image).

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

```
def mean(sub_img):
    sum=0
    for i in range(8):
        for j in range(8):
         sum+=sub_img[i][j]
    return sum/64;
```

2. *Standard Deviation*: The second color moment is the standard deviation, which is obtained by taking the square root of the variance of the color distribution.

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

Ei is the first color moment

3. *Skewness*: The third color moment is the skewness. It measures how asymmetric the color distribution is, and thus it gives information about the shape of the color distribution .

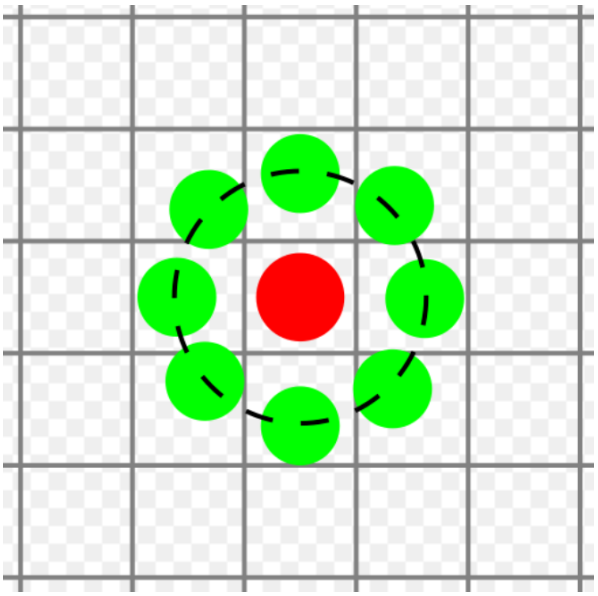
$$s_i = \sqrt[3]{(\frac{1}{N}\sum_{j=1}^{N}(p_{ij} - E_i)^3)}$$

Each block will have 3 colormoment values so the dimensions of final color moments is 64*3. The below snippet is the color moment of first eight 8*8 blocks of the above image

0.576777	0.162762	-0.559286
0.768995	0.0176976	-0.723382
0.799694	0.014497	0.409724
0.876593	0.0253102	-0.54344
0.840012	0.0338709	-0.192571
0.797488	0.0172109	-1.26876
0.794853	0.02054	-0.999294
0.524387	0.168204	-0.177655

2. *Extended Local Binary Patterns* (ELBP): LBP is derived from a general definition of texture in a local neighbourhood. It is used to calculate image texture.
The basic workflow of LBP is shown below:

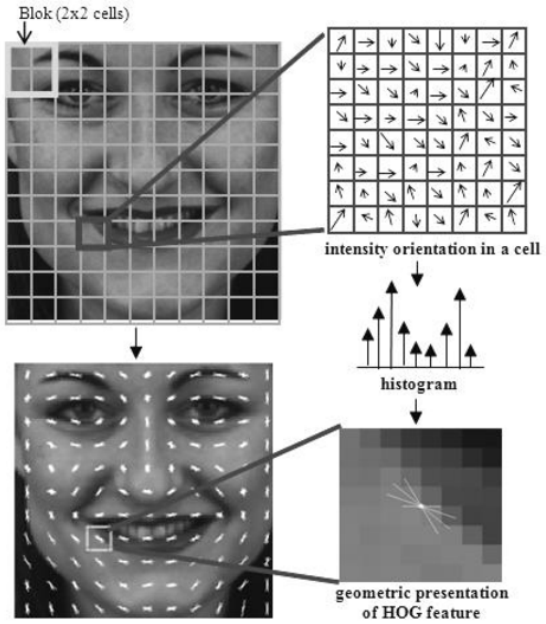
1. The image is divided into different block sizes.
2. Each pixel value is compared to its neighbor pixels in either clockwise or anticlockwise direction.
3. If the center pixel value is greater than the neighbor pixel then allocate 0 else return 1.
4. We then normalize these created values by storing them in a histogram.
5. These Normalized values are then combined for the complete image to give LBP values.



```
def Get_elbp(img):
    lbp = feature.local_binary_pattern(img, 8, 1, method='nri_uniform')
    results = []
    for i in range(8):
        for j in range(8):
            sub_img = lbp[i*8:(i+1)*8, j*8:(j+1)*8]
            (hist, _) = np.histogram(sub_img, density=True, bins=59, range=(0, 59))
            results.append(hist)
    return np.array(results).flatten().tolist()
```

The above function is used to return the elbp values of an image. 59 bins are used in histogram.

3.*Histograms Of Oriented Gradients:* This model is a feature descriptor with the purpose of object detection. The occurrences of gradient orientations will be counted in localized segments of an image.
It might give better results when compared with other models since it considers both inclination & magnitude of image.



The basic procedure in HOG is given below:

1. The gradient vector is calculated for each divided cell of a image.
2. We then calculate the slope and magnitude for pixels.
3. After that we take a histogram of 9 bins. And pixels slope & magnitude will decide into which bin it would get allocated.

```
def Get_hog(img):
    return feature.hog(img, orientations=9,
        pixels_per_cell=(8, 8), cells_per_block=(2, 2), block_norm='L2-Hys')
```

Task-2

Goal: In task-2 we return feature descriptors based on all the models for each image in a folder.

Approach: I iterated through .png file in the given folder, convert the image into array form and then pass it to functions of all 3 feature models.

After that just store all of them combined in a data structure which can be then dumped into a json file.

```
def task2(path):
    features=[]
    os.chdir(path)
    for file in glob.glob("*.png"):
        img_arr=convertpng2array(file)
        cm=colormoments(img_arr)
        elbp=Get_elbp(img_arr)
        hog=Get_hog(img_arr)
        feature={"id": file, "color_moment": cm,
                 "hog": hog, "elbp": elbp}
        features.append(feature)

    return features
```

For example, features is returned from task2 function.

If we want to access color moments of image-0.png

Then we just search in this way :

For f in features:

    if(f[‘id’]==’image-0.png’):
        return color_moments=f[‘cm’]

We first search for the image id then access the color moment of that feature.

```
[
    {
        "id": "0_0.png",
        "color_moment": [
            [
                0.5768995236139745,
                0.16421805648954035,
                -0.5530019540311799
            ],
            [
                0.7696691313758492,
                0.017866851677019034,
                -0.6906328121026809
            ],
            [
                0.8011029530316591
```

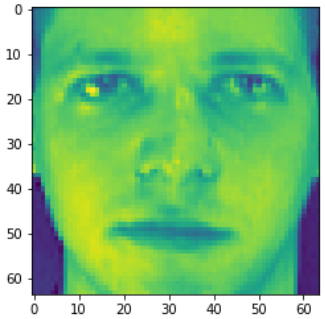

the snippet above is of features that are stored in json file.

Task-3

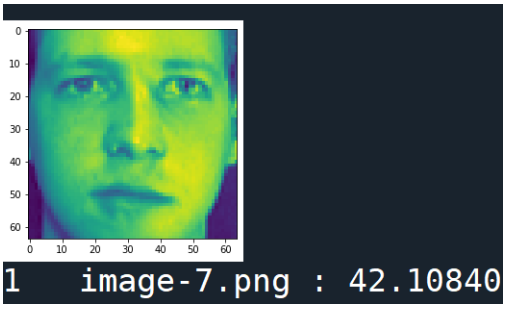
Goal: Retrieval of most similar images with the given image needed to be done in this task based on a given model.
Approach: I am passing the given folder to task2 function firstly to get features

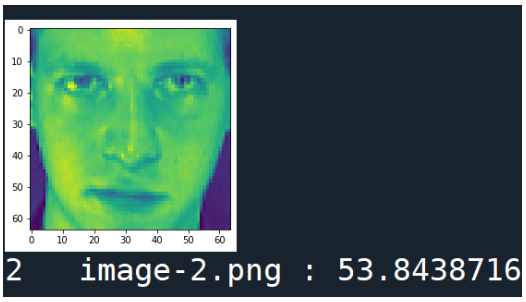
Color Moment: Metric used is Manhattan distance. Input & outputs are from Set1. These are the most similar images in the set1 for image-0.png. Manhattan has given perfect results for Color Moments. It appears that color variations are perfectly getting captured since Manhattan take absolute differences of features.

All other outputs for task3 are stored in outputs folder.

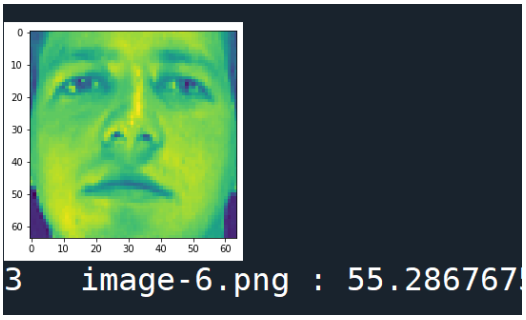


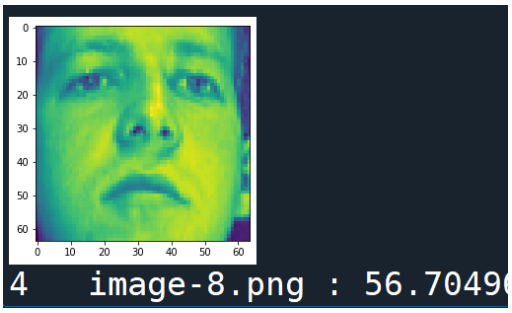
Target Image : Image-0.png





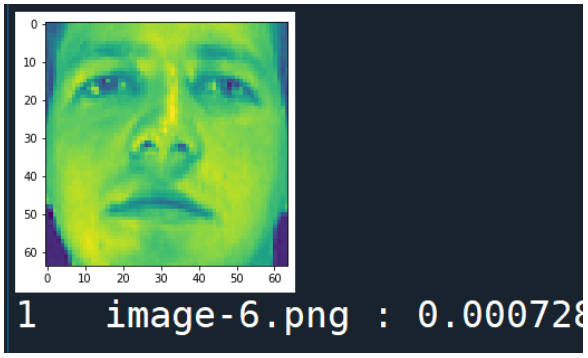
1st Match - most similar 2nd match

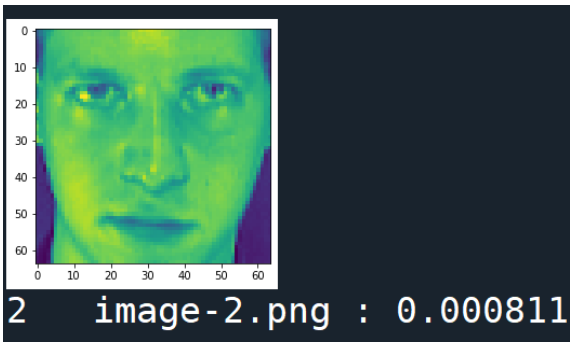




3rd most and 4th most similar matches

ELBP: After trying out euclidean, Manhattan & Earthmovers metrics I have moved forward by selecting Euclidean distance. Since its showing better matching in texture compared other distances mentioned above.





1st & 2nd most similar images for the target image

3rd & 4th most similar images

HOG: The best combination I found for HOG is EarthMovers its giving retrieving the correct person faces even in set2 where images of other subjects are also presnet. Its function in python is available as Wasserstein metric in scipy module. Since it extracts the sharpness very well facial objects like glasses & beard are considered very well.

Top 1 & 2 matches in set2

next most similar matches after 1 & 2

Task-4

Goal: Retrieval of most similar images with the given image needed to be done in this task based on combination of all the models.

Approach: I am passing the given folder to task3 function by making model parameter as 'all'.

In task3 I have made a if condition where it combines feature values from all three models if model is specified as 'all'.

Weightage for each model is as follows:

Color Moments - 1/25

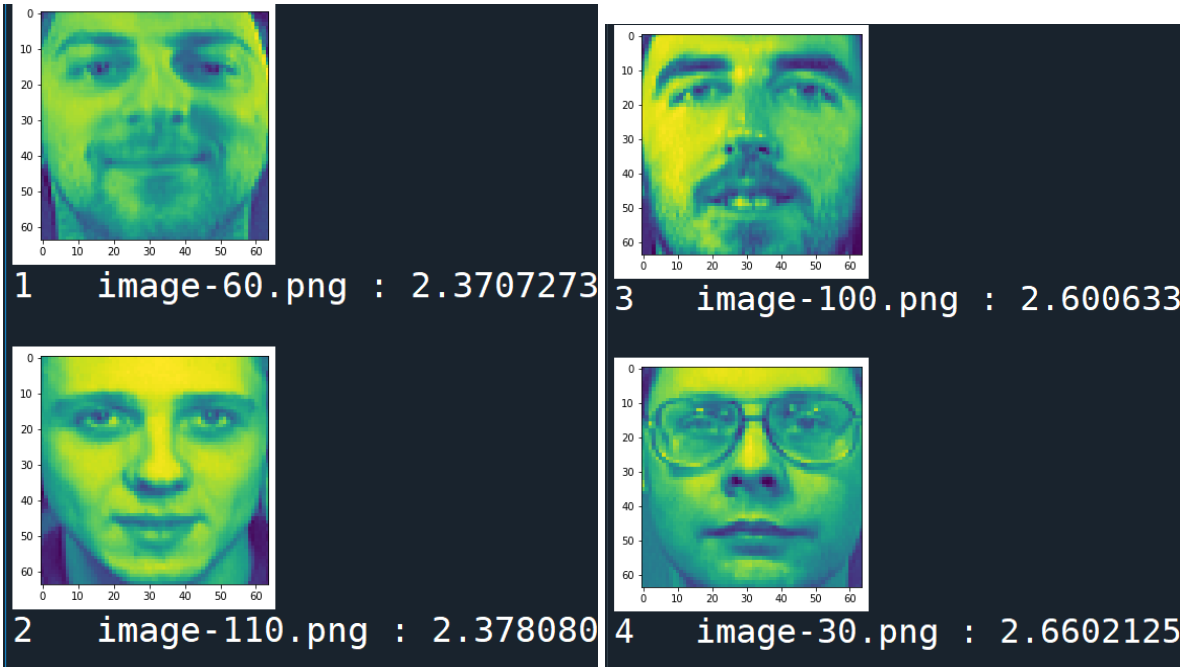
ELBP - 1

HOG - ½

Top 4 images that are retrieved from the combination of all 3 models.

Matching score of each image and contribution of each model is provided below

top4_image_Ids	overall_similarity	ColorMoment	elbp	hog
image-60.png	2.37	2.36	0.0009	0.008
image-110.png	2.38	2.37	0.0023	0.009
image-100.png	2.60	2.60	0.0005	0.004
image-30.png	2.66	2.65	0.0010	0.005



Top 4 images that are retrieved from set3 with mean weightage combination of all three models. Only the faces that are centered without any bending are retrieved since good amount of weightage is given for HOG & ELBP.

System Requirements

I have developed my code in Spyder IDE of Anaconda which has python 3.6 version. The following modules are required to be installed in python.

- PIL
- Numpy
- Skimage
- Json
- Pandas
- Scipy.spatial
- Scipy.stats
- Matplotlib
- Numpyencoder
- Os and global

Related Work

- The paper by Ke-Chen Song and Xu-Zhang mentions the detailed detection applications of LBP and also the rotation invariant patterns in LBP method.
- This helps us understand how a distance metric could vary while comparing the LBP values.

Conclusion

This project was mainly to get us in good touch with Images related feature descriptors & similarity metrics. We have detaily seen how the combination of a feature Model &matching score metric would vary the top ‘K’ matching images result for a target image. Task 3 & 4 assisted me in understanding the concepts of ELBP, HOG in a practical way.

Bibliography

- Dalal, N., and B. Triggs. “Histograms of Oriented Gradients for Human Detection.” *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, doi:10.1109/cvpr.2005.177.
- García-Olalla, Oscar, et al. “Evaluation of LBP Variants Using Several Metrics and Knn Classifiers.” *Similarity Search and Applications*, 2013, pp. 151–162., doi:10.1007/978-3-642-41062-8_15.
- “Hog Person Detector Tutorial.” *HOG Person Detector Tutorial · Chris McCormick*, 9 May 2013, mccormickml.com/2013/05/09/hog-person-detector-tutorial/.
- Pietikäinen, Matti. “Image Analysis with Local Binary Patterns.” *Image Analysis*, 2005, pp. 115–118., doi:10.1007/11499145_13.
- Tyagi, Mrinal. “Hog(Histogram of Oriented Gradients).” *Medium*, Towards Data Science, 24 July 2021, towardsdatascience.com/hog-histogram-of-oriented-gradients-67ecd887675f.