ECSE 415: Introduction To Computer Vision

Assignment 2

Due Date: Thursday, Feb. 10th, 2017

This assignment requires the user to compute SIFT features. If you are using OpenCV 3.00+ you will need to download and install an additional openCV respository, '*opencv_contrib*' at https://github.com/opencv/opencv_contrib. This module contains extra functions that are not distributed as part of the official openCV library. The github link above demonstrates how to install this library. If you followed the steps provided in tutorial #1 you should already have this package installed.

Q1 Harris Corners and SIFT Features [10pts]:

1) In the first assignment you were asked to take 3 images of the same scene from different viewing angles. For this question, you will choose one of these images and perform the Harris Corner Detection algorithm. As per the first assignment, it will be helpful to resize your input images and convert them to grayscale using the built-in openCV functions. A step-by-step guide of implementing this algorithm is provided in Lecture 6 – Image Features slide 46. All portions of this algorithm need to be implemented by hand using nested for-loops. Similar to assignment #1, zero-padding the image will prove beneficial. The output of this algorithm should consist of the input image and a method to display the Harris Corners.

2) You will now choose two of the three images taken in assignment #1 and perform feature matching. The features you will be computing are SIFT features and you are allowed to use the openCV SIFT function to find these features. In addition, you will require a method to compare features, openCV provides a BFMatcher class which allows you to easily find the best matches based on the descriptor vectors and a distance metric. One method commonly used distance metric is a simple sum-of-squared differences or L2 algorithm. The output of this agorithm should consist of the two input images, the corresponding SIFT features, and a method to display the top 50 matches. OpenCV provides a simple method to display all keypoints and a list of matches by using the *drawMatches(img1, kp1, img2, kp2, matches)* function, where *kp1* and *kp2* are pixel locations in image 1 and image 2, and *matches* are returned from the BFMatcher class. You will need to manipulate *matches* in order to display the top 50 results. An example of this matched features function is shown in Fig. 1 below.

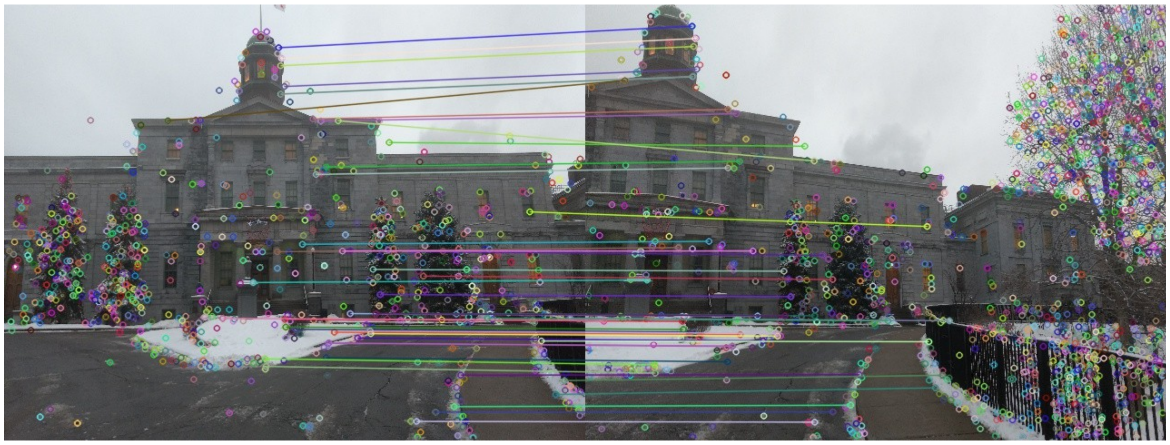

Figure 1 – SIFT feature matching

Q2 Celebrity Face Matching [10pts]

For this question you will be finding your celebrity look-alike. You will need a face image of yourself to use as an input image. I have attached a dataset of 20,000 celebrity images to which your picture will need to be compared. In terms of preprocessing, you should resize all images to (128x128) and convert them to grayscale. To compare face images, you will be implementing, by hand, a Local Binary Patterns (LBP) algorithm. The method we are using is based on the paper by T. Ahonen et al. [1]. The first step is to compute the LBP feature descriptor for your input image. The LBP operator assigns a label to every pixel of an image by thresholding the 3x3 neighbourhood of each pixel with the center pixel value and considering the result as a binary number. The histogram of the labels can then be used as a texture 256-dimension descriptor. Fig. 2 below demonstrates this process. However, this process will only provide us with a global descriptor of the whole facial image. One improvement over this method is to use the texture descriptor to build several local descriptors of the face and combine them into a concatenated global descriptor. In order to do this, the input facial image is first divided into local regions and texture descriptors are extracted from each region independently. For each region, the local histogram is then normalized. The normalized descriptors are then concatenated to form a global descriptor of the face. Fig. 3 shows examples of a facial image divided into local regions.

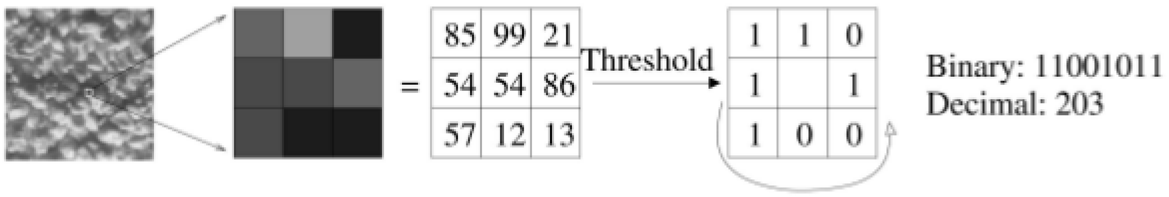

Figure 2 – The basic LBP operator

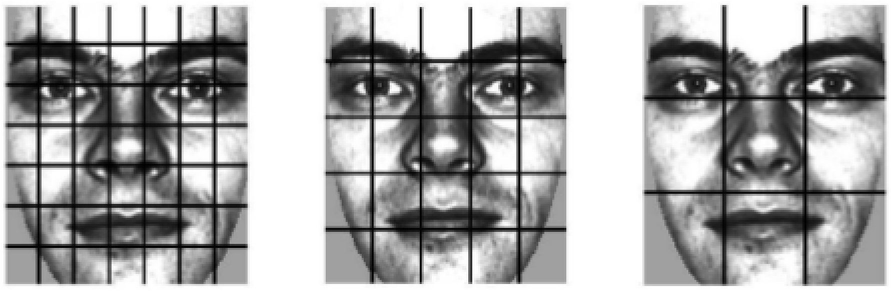

Figure 3 – A facial image divided into 7x7, 5x5, 3x3 local regions

Once you have the descriptor of your facial image, you now need to cycle through all 20,000 celebrity images provided and compute their respective facial descriptors. You must then compare facial descriptors utilizing a sum-of-squared differences, similar to Q1, to find the closest celebrity image.

For this question, your images will have a dimension of 128x128, you will be subdividing your image into 7x7 cells. Each cell will have a dimension of 21x18 pixels (width x height). It is worth noting that these dimensions do not line up completely and the last region will be a small sliver. This is fine since we are normalizing the histograms. The final facial descriptor will then have a dimensionality of 256x7x7. The final output should display your facial image along side of the best-matched image, as shown in Fig 4. You may want to fine-tune these parameters depending on your input image. However, I found these parameters were fairly robust for various input images.

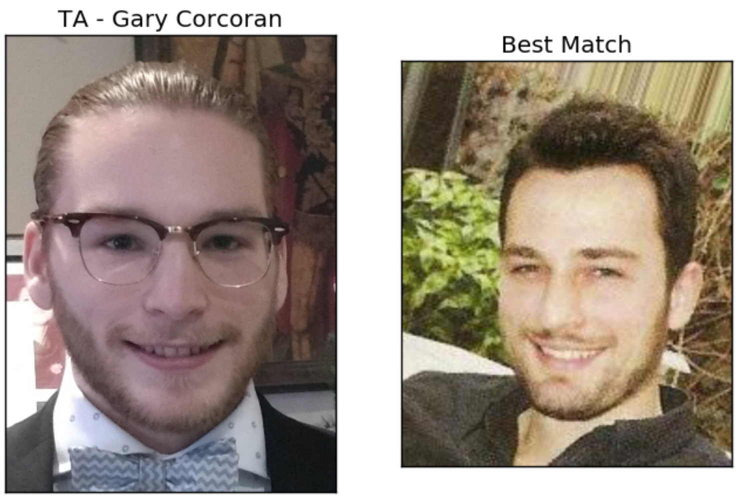

Figure 4 – Celebrity look-alike

Final Comments:
All python code should be written using jupyter. The final submission should contain your .ipynb file, all output and input images, and a short readme file.

REFERENCES
[1] T. Ahonen, A. Hadid and M. Pietikainen, "Face Description with Local Binary Patterns: Application to Face Recognition," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 12, pp. 2037-2041, Dec. 2006.