Programming Challenge – Kopernikus

Challenge summary: In this challenge, an image processing pipeline is implemented to identify and eliminate similar images from a dataset. The dataset contains images acquired in a timely manner, from a typical parking lot. However, for most of the time, being a typical parking lot, there are no changes in the scene, resulting in many images that look similar.

Images are compared with each other to compute the similarity. Initially Gaussian blur is applied to reduce high frequency noise in the images. Then, absolute difference between pixel intensities is computed to identify the change in the scene. Further, contours of changes are identified. Contours are considered to be valid based on their sizes. The sizes of the valid contours of changes, between two images, are added to calculate a score that conveys the degree of difference. If the score is less than a fine-tuned threshold, the image in question is considered as a similar image, and is subsequently removed from the dataset.

Challenge questions

• What did you learn after looking on our dataset?

The moment in the scene (parking lot) in which the images are captured is limited. Hence, large number of similar looking images are present in the dataset. Each image is captured at a unique time stamp. Not all images are of same shape and the datatype of one of the images is invalid. Finally, there is a file of type .DS_Store which is not an image.

In addition, images are captured in both day and night, resulting in wide range of illuminations. If a computer vision application has to be developed for this dataset, the application has to be robust to different kind of lighting conditions. Moment in dark and small objects in low-lit areas in the night could be difficult to identify.

Extra: calibration posters can be noticed on the pillars. From my understanding of the autonomous technology of Kopernikus, these calibration posters help the moving autonomous cars to identify and control the right path in the parking lot.

• How does you program work?

The program contains two modules: imaging_interview.py and kopernikus_solution.py. The imaging_interview.py contains the functions provided by Kopernikus and this file in not modified. The file Kopernikus_solution.py contains the solution to the challenge. Required functions are imported from imaging_interview.py.

The main part of the solution is the class 'SimilarPhotoRemover()'. The class contains three methods, namely: `.load_input()`, `. identify_similar_photos(Gaussian_blur_radius_list_input, score_threshold_input, min_contour_area_input)` and `.remove_similar_photos()`. The object is created and these methods are called in the same sequence, in the main component. The functionality of these methods is as follows:

1. `my_SimilarPhotoRemover(folder_path)`: The constructor of the class takes the path to the dataset folder as input. This path is stored as a class variable.

2. my_SimilarPhotoRemover.load_input(): This method reads all the file names in the folder into a dictionary, where the keys are the names of the cameras, and the values are the images from the respective cameras.
3. my_SimilarPhotoRemover.identify_similar_photos(Gaussian_blur_radius_list_input, score_threshold_input, min_contour_area_input): This method reads two images, current_image and next_image, into the program.
    a. Initially, the current_image is the first image and the next_image is the second image in the dataset.
    b. Both the images are subjected to preprocessing, using the preprocess_image_change_detection() function.
    c. The similarity between the current_image and next_image is calculated using the compare_frames_change_detection() function.
    d. If the score is less than *score_threshold_input*, the name of the next_image is added to a list of unwanted images. Else, we hold the current image as it is, and iterate the next_image onto the next image in the dataset and repeat the process from step a. In other words, we update the 'current_image' only when we find the new unique image.
4. my_SimilarPhotoRemover.remove_similar_photos(): This method iterates through each file name in the unwanted_images list, and removes those images from the dataset folder.

• What values did you decide to use for input parameters and how did you find these values?

1. gaussian_blur_radius_list: Experimented with multiple combinations: [3, 5] [5, 7] [7, 9] [9, 11]. For [3, 5] the blur is very minimal, yet the numbers on the number plate of the car in image c23-1616744495180.png are still clear. At [5, 7], the image is more blurred, yet the numbers on the number plate are still readable, and the edges of the car are perceivable. At [7, 9] and beyond, the numbers start merging into clusters and the edgs of the car start to fadeaway. In order to reduce the high frequency noise in the image, yet preserve the characteristics of the objects in it, **[5, 7]** is selected as the optimal gaussian_blur_radius_list.
2. Min contour area: Identified a couple of consecutive images [c23-1616744495180.png, c23-1616744837702.png], whose difference is a small car in the dark, and Considered this as a thresholding edge-case. When the contours are computed for this car, the whole car is identified as a group of multiple small contours, with the smallest contour being of size 16 and the largest being 920. However, majority of them are above 50. Hence a **Min contour area of 50** considered.
3. Score: Further, for the above case, the score is obtained to be 1524. Considering a safety factor of 2.0; the score 750 (that is approximately half of 1524) is considered as the threshold.
    a. Note: for majority of the other cases, the score is in the orders of few thousands. However, it is better to allow some similar images, at the benefit of not losing images where objects are very small and attribute to change in the scenario.

• What you would suggest to implement to improve data collection of unique cases in future?

1. **Algorithm:** The algorithm now implemented identifies the change in illumination of the scene also as a difference. Deep learning algorithms such as object detection algorithms, or time-series based motion detection algorithms can be used to identify unique scenes, by ignoring illumination changes.
2. **Sensor:** A Time of Flight camera or a Flash LiDAR sensor can be used, instead of an RGB camera. This way, only when there is a difference in the 3D scene, the scene can be considered as a unique

scene. This solves the problem with RGB images, where, changes in illuminations and shadows also reflect as difference between images.

• Any other comments about your solution?

My solution is based on an assumption that; this algorithm is implemented to retain or discard an image captured in the real time. Hence, the algorithm compares the last unique image with the latest image at hand.