Programming challenge for Machine Learning / Computer Vision Engineer role

31 October 2021

Name: Rishabh Raj
Email: rishabh.raaj17@gmail.com
Contact: +49 1630152419

Tasks:

1. **What parameters you decided to use for the provided example dataset?**
   The parameters used for the provided example dataset are:

   - min_contour_area: 500
   - black_mask: (0, 30, 0, 0)
   - gaussian_blur_radius_list: [21]
   - duplicate_threshold/score_threshold: 550
   - contour_count: 6300 (new parameter)
   - global threshold for cv.threshold: default 45

2. **How you found these values?**
   To find the best values for the parameters, I chose to perform a grid search among the possible values. To narrow the search down, I empirically hand picked the boundaries for the search by experimenting over wide variety of images and observing how the changing parameters influence the similarity score. One can also use random grid search but I opted for complete grid search to come up with the best set of parameter values as the amount of data I had to process allowed for this. I will now talk about each parameter separately:

   - **black_mask**: The idea behind this parameter is to mask out areas of image which don't contribute to the duplicate detection. I picked (0, 30, 0, 0), which masks out the ceiling area as it remains static all over the dataset. This helps in reducing some of the computational effort in terms of both time and resources. I avoided masking out any other area where we might need to detect change in the future even though there was none in the provided images. I tried to keep my solution as generic as possible while still tuning the parameters to the specific case. The cv.dilate used causes dilation at the borders of this masked area.

- **min_contour_area**: This parameter decides how much of a local change in any area of image is interesting to decide whether the image is a duplicate. We calculate the contours of the thresholded difference image between the two frames in consideration for duplicate detection. Now, a very small contour area is usually an indication of noise like background object motion or pixel intensity changes here and there. I empirically chose 500 as the parameter value to focus on contours that are not very small, but also keep it not too large to miss significant contours. There is always a trade off in setting such a parameter between the size of contour and the actual motion. It happens that some intensity changes only contour areas are considered with such a threshold, but we don't want to be too strict to miss some images that are actually not similar but yet are removed due to a high threshold for this parameter. I found the optimal paramater by trying out a bunch of different values for different edge cases present in the dataset and then making a observant educated guess that works for all cases in general.
- **gaussian_blur_radius_list**: This parameter is used to smooth out the noise from the images being compared. This helps in removing the small noisy contour detections later in the pipeline. The sigma for the kernel of gaussian blur must not be very small as a bigger sigma helps remove high frequency noise - caused due to camera sensor or some slight illumination changes - whereas a smaller sigma can leave a few noisy contours. I tried out a bunch of different values for different edge cases and chose a best value that works for every case in general. I tried out both ways:
  - A successive application of filter with different sigmas, like [13, 9, 7, 5]
  - A single big sigma, like [21]

  Both of them has a similar effect as theoretically applying successive sigmas is equivalent to a bigger single sigma. I chose to use a single big sigma, to save some computations.
- **duplicate_threshold/score_threshold**: This parameter decides whether the two images are duplicate of each other or not. A small value indicates that two images are same, while a larger score indicates changes. For this, again, I experimented on some edge cases present in the dataset to finally chose a value of 550, that detects and keeps all the different images in the dataset. While this threshold value helps to remove duplicate images, it also allows us to keep images that are not too different, one such case is where in one image the car has its doors closed and in other its open. On the other hand we fail to remove some images which are duplicates but are not considered duplicates due to significant illumination changes like shadows, sunset, sunrise, or movement of the sun.
- **global threshold for cv.threshold**: The default value 45 for thresholding the difference image into a binary image, proved to be a very nice estimate. So I kept it the same. This value can counter the effect of some illumination changes, like illumination due to movement of the sun, and artificial lighting very well.
- **contour_count**: This is a new parameter introduced by me. We get the contours detected in the difference image. The idea is that on two very similar images the count of total pixels contributing to all the contours should be low; while on two images with drastic changes, a lot of pixels contribute to the detected contours. Since we are using global thresholding, we fail to detect duplicates where the change is only due to illumination changes, an example

could be the sunlight coming inside the parking lot, which causes a lot of contours to be detected. A very large count of pixels contributing to contours typically means a lot of illumination changes. By setting a threshold of 6300, we can avoid images where the count of pixels contributing to contours are due to large illumination changes. I found the optimal value by experimenting on a lot of samples and edge cases. There obviously is a trade off and it fails when there is an actual considerable change in the scene not merely due to illumination changes. But the selected value proves to be good enough to include all the distinct images and filter out some illumination only changes images.

3. **What amount of duplicates script found with these parameters?**
   The script was successful in removing 339 duplicates using the aforementioned parameters. The script was successful in keeping all the different distinct images present in the dataset even when the change was very small, for example where only a car door is opened, or a person comes in the scene. On the other hand it fails to remove some images where changes are only due to illumination. This is mostly due to the choice of thresholding used. An adaptive thresholding mechanism can help filter some of them out. I tried to test this and even though it was not the best solution, nevertheless, it helped remove some illumination based duplicate images.

4. **What you would suggest improving to make data collection of unique cases better?**
   Given the size of the dataset provided, it is easy to go through each of them and decide the pairs interesting to find a good set of parameters. But this can get difficult very easily given the frame rate of modern cameras and the huge size of footage one usually has in a real-industrial setting. There are a few ideas that came to me while I was solving the problem. I will enumerate them below:

   - This current script and the choice of methods is a good starting point to detect similar images. We can perhaps use adaptive thresholding which can take care of the illumination changes better than the global thresholding counterpart. Gaussian smoothing of images helps remove noise, we can also use erode from OpenCV to help remove some of the noise, but it is not required in conjunction with gaussian smoothing. Smoothing too much can cause problems detecting objects where the background and the object color is very similar. One such case in the current dataset is with black cars in low light frames.
   - One idea could be to project the images in a lower dimensional space by using some handcrafted features like HOG, SIFT/SURF descriptors or by learning a space using neural networks. Then perform clustering in this space, it can then group similar images together. Using hierarchical clustering on top of it, we can even preserve images with small changes. One very famous architecture for this is Siamese networks using triplet and pair loss. Deep metric learning methods can surely help us to remove more similar images. This can also help to reduce the search space by a significant amount. This idea can be classified as a Neural Search technique, which can even handle multimodal data. For the current dataset size, all of these are overkill, but to create a training set with much larger data dumps, this could be more effective than such basic approach.
   - Another idea could be to perform object detection to detect and count objects in the sequence of images and filter them on the basis of similar or different objects. This might also be more suitable for detecting occluded cars in the far back.

- Another idea could be to use canny edge detector and use 2D chamfer distance metric to detect similarity on the detected edges. If there are new edges, it could mean a new object is in the scene and we can say the images are different.
- Data augmentation either from real distribution or simulated distribution can also help.
- We can use metrics like SSIM and/or MSE to measure the difference in the images and remove duplicates or near duplicates.

5. **Any other comments about imaging_interview.py or your solution?**
   The script does a very good job in removing duplicate images in general. But it is not robust and fails to tackle illumination changes and hence fails to mark them as duplicate. Nevertheless, it is a good starting point in duplicate detection and better solutions can be built on top of it. It can be improved using the ideas presented in the question above. My solution strictly uses the provided functions and hence inherits the drawbacks of the core algorithms. Using an adaptive thresholding or other techniques like clustering or similarity estimation using metrics like SSIM or MSE can help improve the results. Given the scope of the task I adhered to the instructions of the challenge.