By: Rahul Jajodia
Date: 05 July 2023

What did you learn after looking at our dataset?

The dataset has a lot of variation in terms of 1- image resolution, 2- lighting conditions, 3- field of view, and, 4- objects present in the scene. The dataset also has some 5- distortion present but is missing metadata to correct for distortions

1- c10 has two different resolutions for images, namely 640x480 px and 2688x1520 px, which is again very different from the resolutions used for the other three cameras at 1920x1080 px. C21 also did have a few outliers in this category, which I ignored and deleted from the dataset before running the program. Namely, c21_2021_03_27__12_53_37.png which has a resolution of 10x6 px, c21_2021_04_27__12_44_38.png which has a resolution of 1200x675 px, and c21_2021_04_27__12_04_38.png which has a resolution of 1100x619 px. C21_2021_03_27__12_53_37.png has a very small resolution to get any meaningful results, whereas the other two resolutions of 1200x675 px and 1100x619 px were unique to the dataset. I could have worked around it with image resizing and padding but tuning the parameters to these specific images seemed excessive for this task. There was also a corrupt image c21_2021_03_27__10_36_36.png which was purged.

2- Many images are overexposed due to sunlight (c10-1623913758221.png) and other light sources such as the exit sign (c23-1616704118786.png) or ceiling lights (c20_2021_03_26__05_01_37.png) which make the deduplication problem a challenging task. For the images overexposed due to sunlight, the problem is especially challenging because the light source is non-static. This leads to the shadows and glare being far apart from each other in sequential frames, especially if the images are temporally far apart. For the static light sources, image subtraction should take care of this even if the frames are temporally far apart. The only exceptions are the transition images where the lights are off in one image and on in the other, and vice-versa.

3- The field of view is also very different between camera views. For example, c10 has a much narrower field of view than c20 or c23. This means that objects that occupy a certain image area in c10 could occupy a much smaller area in the images of c20 or c23 due to being much further from the camera. Consequently, the min_contour_area parameter would thus have to be much

smaller for c20 and c23 than c10. Having depth information coupled with the camera parameters could help curtail this to some extent.

4- The scenes also provide a good reflection of real world scenarios. There are often pedestrians (c20_2021_04_27__12_50_19.png) and objects (c20_2021_04_27__11_18_28.png) present in the scene. Some are closer to the camera, like in c20_2021_04_27__12_33_10.png while some are much further back in the view, such as in c20_2021_04_27__12_21_45.png. There are also cars frequently entering and exiting the parking lot. With one of Kopernikus automotive problem statements being autonomous parking, I believe this dataset provides good insight into the daily facets of a parking lot. It thoroughly familiarises me, as a candidate, to the challenges faced in solving such a complex problem.

5- Lastly, there is prominent distortion in the images, particularly barrel distortion. This can be fixed with camera parameters if one could encode it to the metadata during dataset creation.

Overall, the dataset provides a good sample of a real world scenario with varied scenes, unknown camera parameters, distorted and noisy images and diverse lighting conditions.

How does your program work?

The program follows the pipeline as illustrated in figure 1.

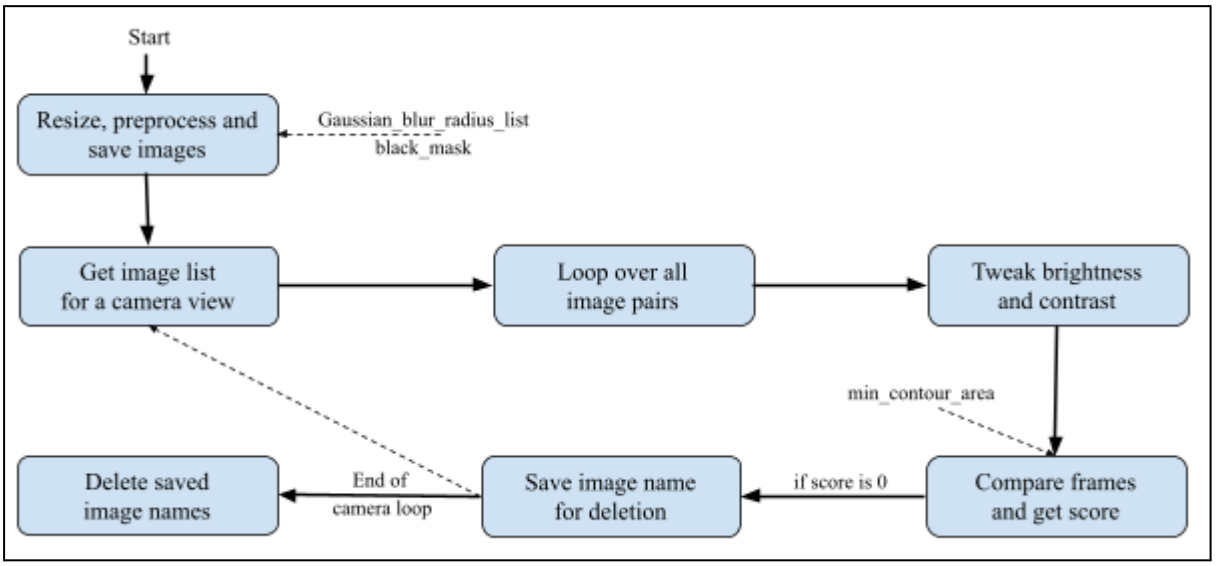

Figure 1: Pipeline for deduplication workflow

At the start, all the images are read in grayscale, resized to a size of 640x480 px, preprocessed and saved. Using the same resolution for all images makes comparing the images much easier later in the program. Preprocessing and saving the images in this stage saves a lot of computational power during the scoring phase, making the program run much faster.

Next, camera views are looped over using a global variable named *CAMERA* that saves the camera ids. This is then used to filter out images belonging to a particular camera id.

After filtering out images using the camera id, we start from the 0th position and compare it with each subsequent image for de-duplication using a nested loop. At each stage of the loop, the pairs are passed to a brightness and contrast tweaker to make the histogram distribution more even. An *alpha* value of 0.7 and a *beta* value of 30 was picked for this operation after some experimentation. After this, a score is calculated using the given algorithm and a min_contour_value. If the final score is a zero, the image from the inner loop is marked as a duplicate of the image from the outer loop. The image name is saved in a list called *dupli_list* for later use. In every subsequent iteration of the loops, if an image, which has already been marked as a duplicate, is encountered, it is skipped.

To explain my motivation behind this choice, please consider this thought experiment. Suppose one of the images has a car at position x=100 and y=100. A second image has the same car at x=150 and y=150. Let us assume the 2nd image is marked as a duplicate of the first image using our chosen parameters. Now we introduce a third image with the same car at x=200 and y=200. If we score the 2nd

and 3rd images, the 3rd image will definitely be marked as a duplicate of the 2nd and filtered out. However the 3rd image may not be a duplicate of the 1st image as per the parameters chosen. To avoid this, once an image has been marked as a duplicate, it is not scored with any other images and is skipped instead. This ensures the 3rd image remains in the dataset after deduplication.

Once all possible image pairs have been scored across all cameras, *dupli_list* is used to delete all marked images from the original dataset and the resized images are deleted.

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

All images are resized to the same resolution. So it seemed like a good choice to use the same blurring radius and min_contour_area for all images across all cameras. The only parameter that differs is the black_mask as it is purely view-dependent

Camera ID	*blur_radius*	*black_mask*	*min_contour_area*
c10	[5,5]	(0,13,0,0)	500 px^2
c20		(0,28,0,0)	
c21		(0,30,0,0)	
c23		(0,35,0,0)	

Table 1: Hand-picked parameters for different resolution images in different camera views. *blur_radius* is *gaussian_blur_radius_list*

All the parameters were handpicked after extensive testing for the use-case of autonomous parking. As an extension to the given task, one would want to segment out objects from deduplicated images to allow for object segmentation and recognition. Large blurring kernels, or masking the sides can impede that. The parameters were selected with that in mind.

Motivation:
The motivations for the *gaussian_blur_radius_list* and *black_mask* are almost the same across all scenes.

gaussian_blur_radius_list:
Blurring is generally done to denoise the images. The chosen blurring kernels achieve a nice balance of denoising the image while not blurring the image too much so as to lose valuable segmentation details for thresholding. Ideally, larger images allow for bigger blurring kernels without sacrificing segmentation details.. However for smaller resolutions, such as 640x480 px, two 5x5 kernels for Gaussian blurring seemed sufficient.

black_mask:
The main goal with the masks was to hide the ceiling beams and lights while not hiding the sides. The sides are not masked because it can provide valuable information such as open parking spots in c20, approaching vehicles/pedestrians in c10 and c21,

and a mix of both in c23. This information is important as having a precursor to an approaching object can help avoid collisions. Knowing which parking spots are available are also vital for autonomous driving. As such, the provided scenes did not allow for masks on the sides or at the bottom. A plausible situation for not masking the ceiling would be if it was undergoing maintenance, but even then, we would still be able to see at least a ladder or other equipment on the ground.

min_contour_area:
This was the trickiest parameter to hand-pick. While a small min_contour_area would make marking duplicates really difficult if the images provide noisy thresholding segmentations, a large min_contour_area would mark non-duplicate images as duplicates. Ideally, a different threshold should be chosen for each view. However, for this particular dataset, a value of 500 px^2 allowed to filter out most small thresholding artefacts after two iterations of dilation while preserving objects large enough to obstruct autonomous parking. A resolution of 640x480px reflects that one must be able to detect objects that may appear small but qualify in making two images distinct. The chosen value of 500 px^2 allowed for that in this dataset.

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

Using a single camera to capture deep scenes such as c20 and c23 can be detrimental. It would be much better to use more cameras to capture smaller parts of the scene along the length. Also, accounting for camera failures is important. Cameras should be set up such that adjacent cameras can be used when one breaks down, so as to not have blind spots. An illustration is shown in figure 2.

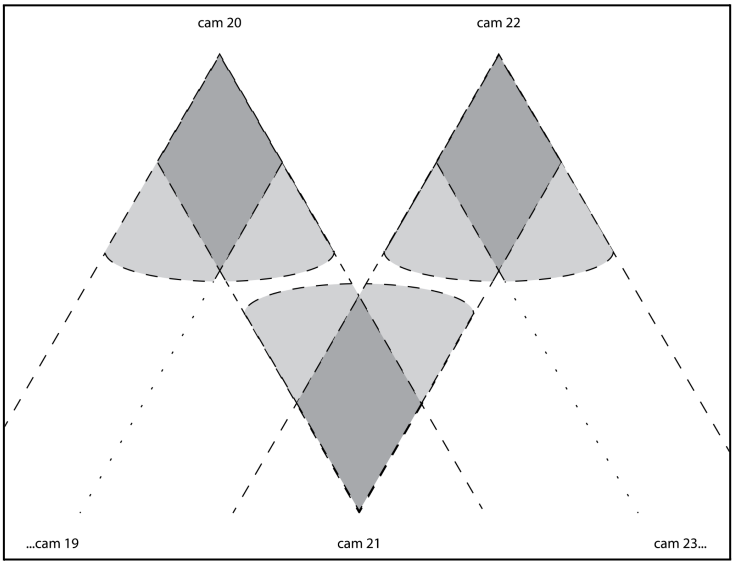

Figure 2: Illustration showing cameras covering each other's views to compensate if a camera breaks down

One of the main augmentations to improve data collection of unique cases in future would be to include depth information and camera matrix, along with images for background subtraction. While some views did have images with empty scenes which could be treated as background, some did not. Further, undistorting images can also help improve the dataset, although this requires distortion coefficients. A robust way to calculate distortion coefficients could be to stick ArUco or chessboards in strategic locations in the background.

Any other comments about your solution?

Originally I approached the problem as a temporal one. I was comparing each image only with the next one in the temporal space, sequentially, to check for duplication. I had misunderstood the task and thought the images being named using timestamps meant that I would need to deduplicate the timeline and not the dataset itself. After some feedback, I realised my mistake and fixed my approach.

During the time I spent working on my original approach, I also experimented with many algorithms to get a better solution when subtracting one image from another, particularly for the images with overexposed pixels. However the problem seemed too difficult to solve in the given time constraint. I tried adaptive thresholding, otsu's binarization, different combinations of erosion and dilation. Ultimately, adjusting the brightness and contrast of the image using an *alpha* value of 0.7 and a *beta* value of 30 to get a more distributed image histogram combined with the given thresholding seemed to give the best results. The most difficult images to work with were those with sunlight. Large glares in the image made it difficult to filter out similar looking images as duplicates with the chosen *min_contour_area*. However, picking a larger *min_contour_area* would have had other detrimental effects and filtered out images that were indeed not duplicates. To this end, I considered having more false positives a safer approach than having false negatives. I think having background images for all view cases to background subtract would definitely help, particularly in the scenes with heavy or very low lighting. I would also like to experiment with gamma correction by using a low gamma value for bright images and high gamma value for darker images. Ideally one should be able to distinguish between such scenes by looking at the image histograms. It should help in achieving a more robust background subtraction and thresholding. Having camera parameters or depth information would also make contour filtering more robust.

Please note that I have not included error checking code in many places. This is purely due to time constraints.

Thank you.