Kopernikus Automotive- Programming challenge for Perception team member

Submitted by: Rakesh Reddy Kondeti

July 11, 2023

1 What did you learn after looking on our dataset?

The given dataset consists of data captured from 4 surveillance cameras- [c10, c20, c21, c23] in the parking garage. For any given camera, the background is always stationary. So, for a given camera, only lighting conditions and objects vary with the background being constant.

Having said that, for a given camera, most of the frames are (near) similar to each other. This kind of data cannot be directly used for training machine learning algorithms. Because all these very similar images induce bias into the dataset thus hurting the overall generalizability of the network. Therefore, identifying duplicate and very near similar images is important.

2 How does your program work?

To run my program, the *main.py* has to be executed and for the same, the following arguments are required from the user.

- *path_to_image_directory*: path to the image directory. Any non-readable and corrupted images are safely ignored.
- *path_to_results_directory*: path to the output directory. All the non-duplicate image frames are saved here.
- *minimum_contour_area*: Thresholding value for the contour areas for them to contribute to the overall score.
- *similarity_score_threshold*: Once the overall score is computed and it is greater than this threshold, then it is considered to be a dissimilar frame when compared to the previous frame.

Once we have all the above-mentioned values, the *main.py* script can be executed as follows:

```
python ./main.py <path_to_image_directory>
                 <path_to_results_directory>
                 <minimum_contour_area>
                 <similarity_score_threshold>
```

For example, if the path to the image directory and result directory are *D:\dataset-candidates-ml \dataset* and *D:\dataset-candidates-ml \dataset \results* respectively. The minimum contour area to contribute to the overall score is 5000 and the threshold value of the score is 50000. Then the main.py can be executed as:

```
python ./main.py D:\dataset-candidates-ml\dataset
D:\dataset-candidates-ml\dataset\results 5000 50000
```

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

The most important and critical parameters are *minimum_contour_area* and *similarity_score_threshold*. I personally found no other deterministic way to find these values. So I followed a trial-and-error method.

The parameter *minimum_contour_area* decides whether the contour area of a specific region can contribute to the overall score of that particular frame when compared to the previous frame or not. For instance, I found that contour areas of small dots or blobs can range anywhere between 16 to 50. However, such small regions do not have any meaningful structure in them and considering them for computing the overall score does not make sense! Through running the code iteratively, I found out that areas greater than 5000 have some significant structure and hence considered this value as the *minimum_contour_area*.

Similarly, *similarity_score_threshold* decides whether the given two frames are similar or not. The more similar they are, the lesser the value of the overall score. For example, if the two frames are the same images, then ideally the overall score is 0. If the frames are dissimilar, the score is high. So, the *similarity_score_threshold* decides the frames are dissimilar if the score is greater than this thresholding value. Through the trial-and-error method, I found 50000 gives decent performance overall.

Optionally, I have used Gaussian blur with a radius of 5 to reduce the noise. I have not played around with the Gaussian blurring parameter so much!

Nevertheless, all the above parameters can be further fine-tuned for achieving overall better results.

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

I would constrain the open problem by asking the following questions myself before collecting any unique data from the available data

- In the end, What is the problem we are trying to solve?
- What kind of data do we require to solve the problem?

However, collecting data for simple tasks like image classification or object detection, the approach given in *imaging_interview.py* works. Also, I personally liked this approach as it is very simple yet effective. However, there are a few drawbacks that I could think of:

1. The presented method requires the conversion of an RGB image to a grayscale image. This is a very powerful method as it already suppresses unimportant information like colors while preserving geometric information. But if the application requires us to sample frames by comparing lemons and oranges, this approach would not work, as we are already suppressing color information.
2. I also found that the score obtained following this method is unconstrained or has no upper cap. This would be more intuitive if there exists a score range (like percentage, or 0-1).

I could think of two solutions to handle them:

4.1 Structural Similarity Index Measure(SSIM)

SSIM quantifies image quality with respect to the reference image, by estimating the degradation of structural similarity based on the statistical properties of local information between the two images. If the SSIM value can be expressed in percentages. If it returns 0, then both the frames are completely different and if it returns 100 then they are the same. As two consecutive frames are very similar to each, we can use this method.

4.2 Learning-based Methods

Methods like Dense Vector representation for images (like Siamese network) can be used. However, it requires further fine-tuning of the network for this specific application.

5 Any other comments about your solution?

For the image frames obtained by a specific camera, I am storing all the scores in a list. After processing all the frames for a camera, all the similarity scores are iterated. If the score of a specific frame is less than a particular threshold (here 50000), then it is considered to be a similar frame (with respect to the previous frame) and hence it is not copied into the results folder. Alternatively, one can sample n number of frames with highest scores respectively.