CSE 691: Image and Video Processing

Spring 2020 Assignment 6

Background Subtraction

Lichen Liang

04/02/2020

Objectives

- Understand the Background Subtraction algorithm
- Understand each step of the algorithm and what does it do.
- Know how the variables affect the output.
- Apply BGS to the images.

Method

To run the code, run ***main.m*** and make sure all function files are in same folder.

The output videos and image folders are labeled as 'out[ERODIL]N[number of frames used for training]S[size reduction]. For example, outERODILN50S0.5 means it used erosion and dilation with 50 training images and each image is resized to 0.5 of the original image.

1. I used folder 1 for training data and folders 2, 3, 4, which I later combined, for testing. In the first folder, there are 238 images. I picked N frames for creating the codebook, where N is 5, 50, and 238. First, read all training images and resize each into 50% or 80% of their original size. The reason is to reduce the execution time and the analysis on the size difference will be explained in the next section. For each image, retrieve its RGB values at each pixel and save in 3 different matrices Red, Green and Blue. Then this size for each matrix is MxNxNframes where MxN is the size of the image after resizing and Nframes is the number of images we used for training. Next, call the *createcodebook()* function.

2. For each pixel in the image, and for Nframes, retrieve its R,G,B value to get *xt* and *I* where *xt = [R,G,B]* and *I = R+G+B*. Initially, in the first loop, our codebook is empty, so we have to add *vl* and *aux* to the codebook where *vl = [R,G,B]* and *aux = [I, I, 1, t-1, t, t]* (t is the index in codebook, which is *k* in the for loop inmy code). Both *vl* and *aux* are string arrays and they are paired into one single string array. This array is then saved in our codebook. Starting from the second loop of the Nframes, as long as the codebook is not empty, we have to check whether this codeword is in the codebook already. The number of entries in the codebook is kept by a *wordcount* counter. Retrieve each *vl* and *aux* in the codebook as *Vcurr* and *Auxcurr*, calculate the color distance and brightness with the current *xt* and *I*. For color distance and brightness functions, see point 3 and 4. The color distance returns a value *delta* and brigtness returns a bit 1 or 0 indicating true or false. If *delta* is less than *epsilon*, the detection threshold, and brightness is true, then we have a match in the codebook. Raise the *matched* flag to true and save its index where it was matched. For a matched, we have to update the *vl* and *aux* for this pixel in the codebook. Assume *vl* and *aux* at point *m* in the codebook is *Vm= [Rm,Gm,Bm]* and *Auxm = [Iminm, Imaxm, fm, lambdam, pm, qm]*. The update of *vl* will be

$$vm = [\frac{fm * Rm + R}{fm + 1}, \frac{fm * Gm + G}{fm + 1}, \frac{fm * Bm + B}{fm + 1}]$$

And the update for *aux* will be

$$auxm = [\min(I, Iminm), \max(I, Imaxm), fm + 1, \max(lambdam, t - qm), pm, t]$$

After updating and creating codebook, we have to update the lambda value in each codebook, which is the maximum negative run length value. Again, retrieve the *aux* from each codebook and update lambda(4_{th} position in *aux*). New lambda is calculated by

$$lambdam = \max(lamdam, Nframes - qm + pm - 1)$$

Finally, save this updated *aux* back into the codebook.

For each pixel, it has its own codebook and each codebook is saved in a larger string array *CBfinal* with the size of 1xM*N.

The overall data structure is an 1xMN array of *CBfinal* mapped into a 1xMN array of codebook, and each array in the codebook is an 1x2 that contains *vl* and *aux*.

3. The *colodist()* function calculates delta with input *xt* and *Vcurr*. Let *xt = [R,G,B]* and *Vcurr = [Rm,Gm,Bm]*, delta is calculated as follows. (note this are the same equations as in the paper, just with some simplified variables)

$$X = R^2 + G^2 + B^2$$
$$V = Rm^2 + Gm^2 + Bm^2$$
$$< X, V >^2 = (R * Rm + G * Gm + B * Bm)^2$$
$$P^2 = \frac{< X, V >^2}{V}$$
$$delta = \sqrt{X - P^2}$$

   The code is a direct implementation of the equations and the epsilon value is determined by seeing each delta value in each iteration

4. The *brightness()* function returns a bit for true or false. It uses *Imin* and *Imax* and two values alpha and beta to calculate *Ilow* and *Ihi.* If the norm of *xt* is between the *Ilow* and *Ihi*, then it returns true, else false.

$$Ilow = alpha * Imax$$
$$Ihi = \min\left(beta * Imax, \frac{Imin}{alpha}\right)$$

   This code is also a direct implementation of the equation and condition.

5. Once we have the *CBfinal* codebook, we have to do temporal filter. This step refines the codebook by separating the codewords that might contain moving foreground objects. For every codeword in the codebook, if the lambda value is less than Nframes/2, then add this codeword to the filtered codebook *FilteredC*B.

6. For each image in our testing folder, read in the image and resize just like the training images. Each image is called into the function *BS()* for background subtraction, the output from the background subtraction is a black and white image where white indicate foreground. The output images are saved into a folder where later made into a video.

7. For the *BS()* function extract the xt and I for each pixel from the image that is being processed, and the corresponding Vcurr and Auxcurr from the codebook of that location. Then color distance function and brightness function is called again. If the pixel satisfies the condition(delta<epsilon and brightness is true), then it must be a background pixel and the pixel for output image is set to black, otherwise, set to white.

8. For morphological operations, I eroded the output image with a 2x2 cube then dilated with 4x4 cube to try to remove the noises. It worked to some extent where it removed the noise, but also shrink the foreground object.

Results and Discussion

Speed

The speed of execution is highly dependent on the size of the image. If I use the original size of the image, each frame takes about 2 minutes in the create codebook stage. This is impossible to implement as I increase the number of frames, the program will take few hours to run just for the first function. Instead, I resized the image into 50% of original size, which each image takes roughly about 6 seconds. If I resize the image into 80%, each image will take around 46 seconds. We can see that a slight decrease in image size can contribute to the speed by a large factor. However, the tradeoff is that the output image lose more details such as identifying the exact object of the foreground. For example, a person detected may not look like a person in the output, but rather few very rough white pixels.
The speed for background subtraction is another major contributor to the slowing down of the program. This is because the large number of testing images and for each image, the background subtraction function is called.

Number of Training Frames

This part I only changed the number of frames for the training data and kept all the size to 0.5.

Figure 1. Frame 186, Nframe = 5, S = 0.5

Figure 2. Frame 186, Nframe = 50, S = 0.5

Figure 3. Frame 186, Nframe = 238, S = 0.5

By comparing figure 1, 2, and 3, we can see that with higher number of frames, more noises are being classified as foreground. This is due to more data are being used to build the codebook. Technically, we should use more frames for training, even more than 238 frames. Another thing I want to point out is that in our training set, there are objects moving, so the background model is not perfect. Also, the resolution or quality of the training image is very low (see 'input1.avi') and this may also contributed to the fact that we have lots of noise in the output.

The difference in the objects (the person and the car) is not very obvious, meaning that these are definitely foreground objects.

Size of Image

This part I kept 50 training images and tested sizes of 50% and 80%

Figure 4. Frame 805, Nframe = 50, S = 0.5

Figure 5. Frame 805, Nframe = 50, S = 0.8

Comparing figures 4 and 5, we can clearly see the difference. With the larger image, it has more pixels and details. However, it also had more background noises that are being classified as foreground. It is really hard to say which is better because with a larger image, we can see the foreground object more clear, but it takes longer to process and more unwanted foreground pixels in the output.

Again, the person riding the bicycle and the car may be more clear in figure 5, but its difference is no that great compared to the noise.

Morphological Operation

Usually, if we want to remove the noise and strengthen the object, we would perform erosion then dilation on the image. However, I can hardly say it is a success.

Figure 6. Frame 887, Nframe = 50, S = 0.5, Eroded and Dilated

Figure 7. Frame 887, Nframe = 50, S = 0.5, Eroded and Dilated

Comparing figure 6 and 7, we can see that it is much easier to identify an object in figure 7 which had more pixels. However, in figure 7, it also had more noises. I think there is a large room for improvement in this portion of the assignment. Once morphological operations are success, then we can predict that objects are being shown while the noises removed.

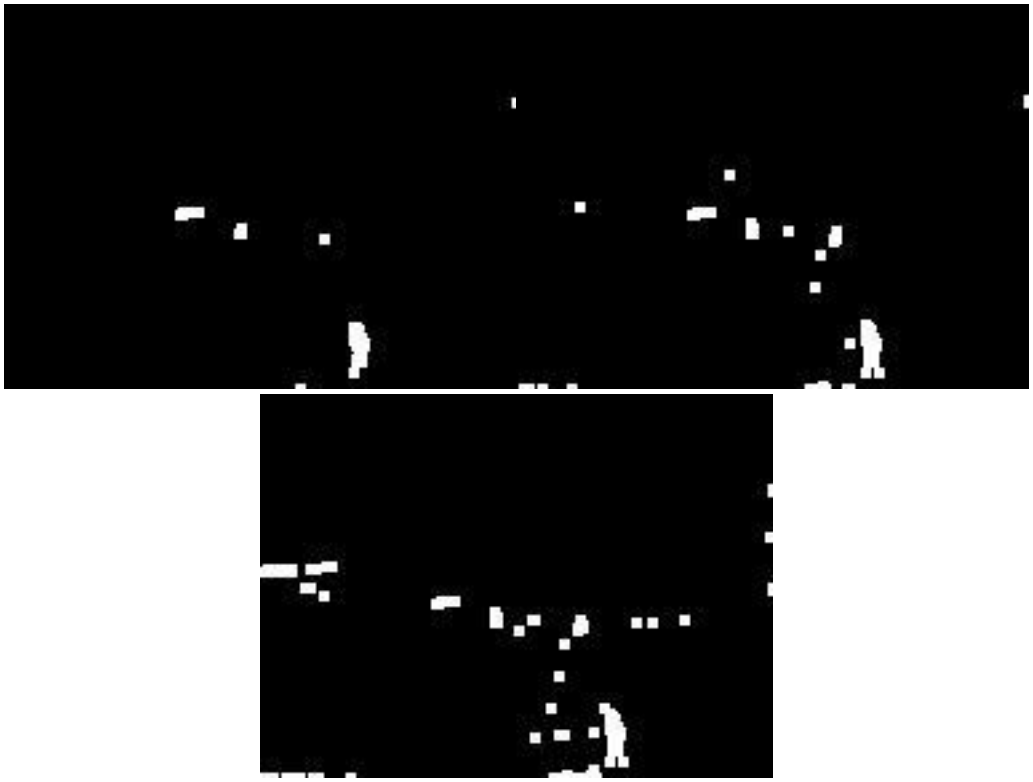

Figure 8. Frame 1393, Nframe = 5(top left), 50(top right), 238(bottom), S = 0.5, Eroded and Dilated

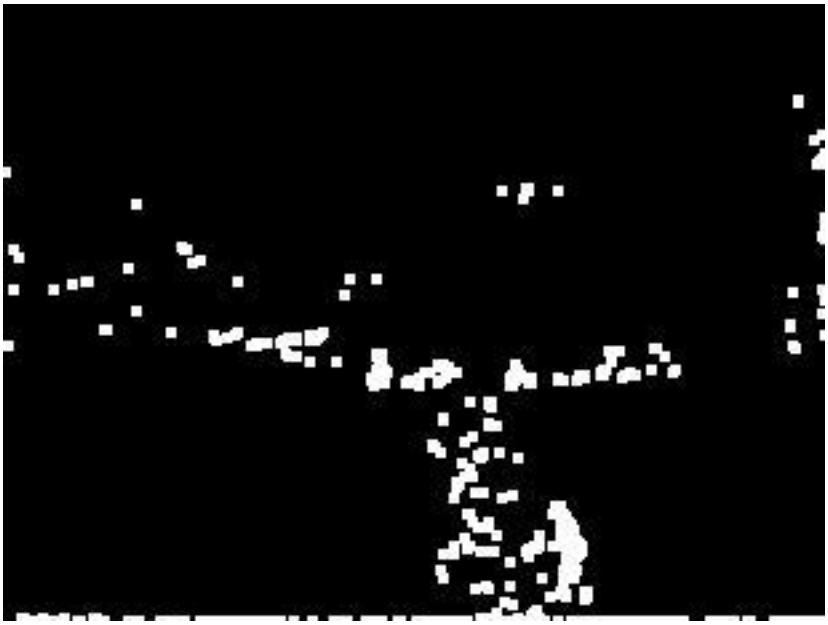

Figure 9. Frame 1393, Nframe = 50, S = 0.8, Eroded and Dilated

Comparing figures 8 and 9, we can see that with Nframes increase, there is slightly more noise but the object gets more visible. With size of the image increase, the same observation is made. This proves the observations we made in the Size of Image and Number of Training Frames section.

5 Images for Detected Objects

Figure 10. Frame 1, Nframe = 50, S = 0.5

Figure 11. Frame 32, Nframe = 50, S = 0.5

Figure 12. Frame 304, Nframe = 50, S = 0.5

Figure 13. Frame 1086, Nframe = 50, S = 0.5

Figure 14. Frame 1535, Nframe = 50, S = 0.5

5 Morphological Images

Figure 15. Frame 1, Nframe = 50, S = 0.5, Eroded and Dilated

Figure 16. Frame 32, Nframe = 50, S = 0.5, Eroded and Dilated

Figure 17. Frame 304, Nframe = 50, S = 0.5, Eroded and Dilated

Figure 18. Frame 1086, Nframe = 50, S = 0.5, Eroded and Dilated

Figure 19. Frame 1535, Nframe = 50, S = 0.5, Eroded and Dilated

Future Improvement

There are many future improvements we can make in this assignment since our results are not perfect. For example, we can try multiple different combinations for thresholds and try to get the optimum result possible. We can also try to improve the speed of the algorithm in multiple ways such as re-writing the code or implement a different and more modern algorithm for background subtraction. The quality of the training data is also very important as it directly affects the result of the testing data. Increasing the number of training images and image size is also very important, but the speed factor just have too much weight on the overall execution. Using multiple morphological operations with different sizes and parameters is another way to improve the output, because erosion and dilation combination is meant to remove the noise and sharpen the image.

Conclusion

In this assignment we implemented background subtraction to test data. Few major observations we can conclude:

- The more number of training images, the longer the algorithm takes.
- The larger the size for each image, the longer the algorithm takes.
- The more number of training images to build the codebook, the better the results.
- The larger the size of each image, the better quality(more pixels) in the output.
- Having a larger size and more training image results in more noise but better noticeable foreground object.
- Noise can be reduced using morphological operations such as erosion then dilation.