COL - 780 Computer Vision - Assignment 1
Dharmisha Sharma(2022MCS2062)
Method - kernel-density based method with decaying weights(Y=2)

Link of one drive containing the generated output video: CV_assign1_videos

Algorithm:

1. Precalculate the Gaussians for the range of pixels, i.e. for values(0 to 255) using the following formula:

$$\frac{1}{\sqrt{2\pi(B_d/2)^2}}\exp\left(-\frac{1}{2}\left(\frac{C_k - x_t^d}{B_d/2}\right)^2\right)$$

   where,
   C_k - ranges from 1.5 to 253.5 (this represents a histogram with 64 bins each of dimension 4 with their centers at 1.5, 5.5, 9.5, 13.5, and so on)
   x_t - intensity of a pixel at time t which ranges from 0 to 255.
   B_d - max(x)- min(x)/n, where n is number of bins, this equals to 255-0/64 = 255/64

2. Take the height, width for the frame i.e. the pixels across the x and y axis.
3. Initialize the base model by taking in account the first frame using the formula:

$$p_t^d(C_k) = \hat{p}_{t-1}^d(C_k) + \frac{1}{G_t\sqrt{2\pi(B_d/2)^2}}\exp\left(-\frac{1}{2}\left(\frac{C_k - x_t^d}{B_d/2}\right)^2\right) \quad ...(1)$$

   For my algorithm I have taken:
   C_k - ranges from 1.5 to 253.5 (this represents a histogram with 64 bins each of dimension 4 with their centers at 1.5, 5.5, 9.5, 13.5, and so on)
   x_t - intensity of a pixel at time t which ranges from 0 to 255.
   B_d - max(x)- min(x)/n, where n is number of bins, this equals to 255-0/64 = 255/64
   G_t - learning rate
   here, **p_{t-1}** = 0, since the model is based on the first frame.

$$G_t = Gain \times \frac{2}{1+\exp(-(cnt-\beta)/\lambda)} \quad ...(2)$$

   For my model I have initialized Gain = 300, β = 100, λ = 20, cnt = 1.
   Learning rate is not considered for the base model built using the first frame.
4. Create a base model for all the 3 dimensions i.e. RGB.
5. Set an initial gradient for all 3 dimensions as one. Gradient will be an array of size (x*y) where x and y are the height and width of each frame respectively.
6. Initialize G_t as per equation (2).

7. Take the frame whose pixels we need to classify.
8. For each pixel, do foreground detection by finding the minimum distance for all 3 dimensions using the following equation,

$$Dist_d = \min_{\forall k}(C_k{}^d - x^d),\ \ where\ \ \hat{p}^d(C_k) > 1/N_d \qquad d = 1,2,3 \qquad ...(3)$$

where, C_k is the same as described above, x^d is the intensity of a pixel for dimension d, N_d is the number of bins, i.e. 64 in this case.
9. After computing the minimum distance for each dimension for a pixel, to determine if it is foreground check if

$$\left(\sum_{d=1}^{3}(|Dist_d|/(1+Grad_{t-1,d})) > \sum_{d=1}^{3} B_d \times \gamma \right) \quad ...\ (4)$$

If equation (4) satisfies, the pixel is foreground, set the value of the pixel as 255(since, we are considering white for foreground and black for background as per the ground truth provided).
10. Update Gradient as follows:

$$Grad_{t,d} = (G_t - 1) \times Grad_{t-1,d} / G_t + w \times |Dist_d| / G_t \qquad d = 1,2,3 \qquad \ldots(5)$$

where, w = 0.3
11. If equation (4) does not satisfy, i.e. pixel is classified as a background pixel, set the value of pixel as 0. In this case, update the gradient as follows,

$$Grad_{t,d} = (G_t - 1) \times Grad_{t-1,d} / G_t + |Dist_d| / G_t \qquad d = 1,2,3 \qquad \ldots(6)$$

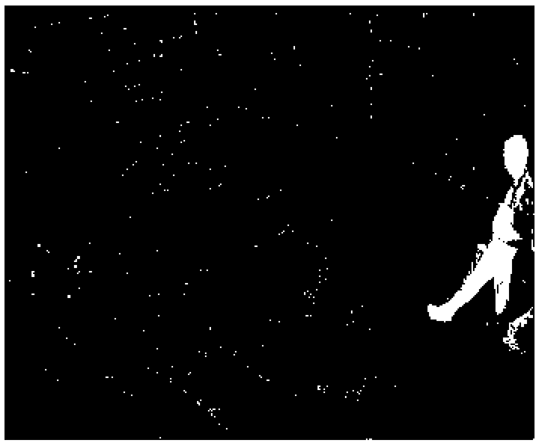
12. Compute a new model based on the values of intensities of each pixel in each dimension, using equation (1), but this time, G_t (computed at step 6) is considered.
13. Update the old model as follows:
    model = old model + new model …(7)
14. To implement it using decaying weights, I took an α=0.95 and modified equation(7) as follows:
    model = (α)*old model + (1-α)*new model …(7)
15. Increment cnt by 1.
16. Repeat steps 8 to 15 for each pixel of a frame.
17. Repeat these steps for each frame with the updated model.

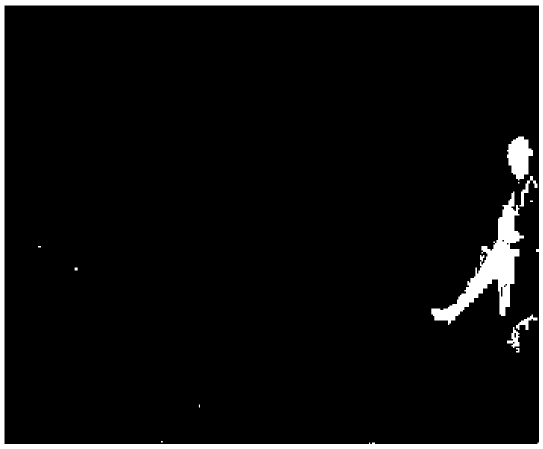
Note: This algorithm also works for grayscale where d(dimension) =1. In that case we’ll have only one model instead of 3.

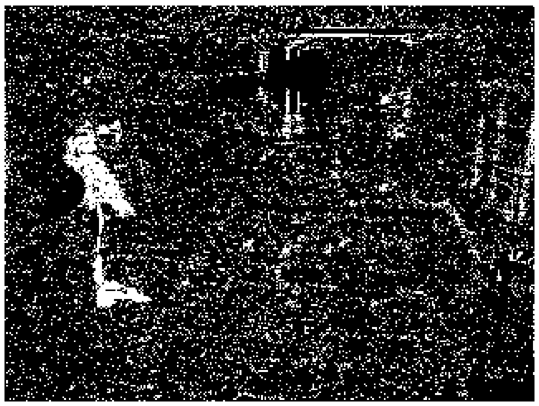
Cleaning:

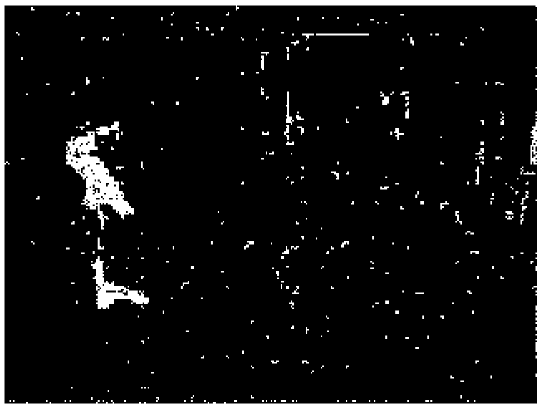
1. Create an original image matrix which is of size (x*y) where x and y are the height and width of a frame respectively. Each cell of this matrix represents the intensity value of the corresponding pixel.
2. Create an integral image matrix from the original image matrix. Each cell (a,b) of this matrix is a sum of all the values of the original image matrix from the box ((0,0),(0,a),(0,b),(a,b)).
3. To compute the sum for a particular box of the integral image the following formula is used.
   ii[a+a1][b+b1]-ii[a+a1][b]-ii[a][b+b1]+ii[a][b] …. (8)
   where, ii : integral image matrix, (a,b): (a,b)th pixel, a1: length of box, b1 = width of box
4. For cleaning my output frames I have taken a box of 3*3 size, i.e. 9 pixels.
5. In this box, I have computed the sum of the box using equation (8), if this sum is greater than some threshold(75% in my case) i.e. more white pixels, I have set the value of each pixel in that 3*3 box as 255 i.e. white. Similarly, if the sum is less than some threshold(25% in my case), i.e. more black pixels, I have set the value of each pixel in that 3*3 box as 0 i.e. black.
6. Repeat steps 1 to 5 for all frames.

   Results:

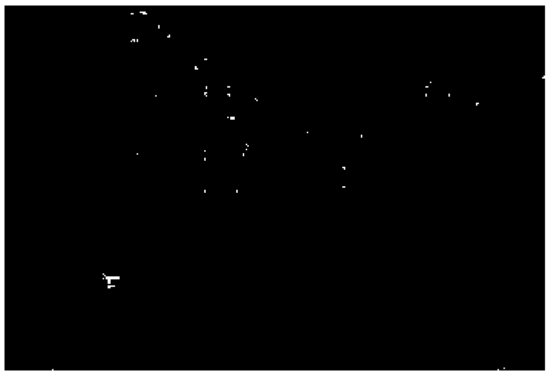

Before


After


Before


After

Before


After

Analysis of Result:

1. **Candela_m1.10:**
   Accuracy : 98.407%
   Confusion matrix:

	Predicted(Black)	Predicted(White)
Actual(Black)	33962387	122088
Actual(White)	442898	954227

2. **CAVIAR1:**
   Accuracy : 98.1320%
   Confusion matrix:

	Predicted(Black)	Predicted(White)
Actual(Black)	56916765	854977
Actual(White)	265132	1928566

3. **HallAndMonitor:**
   Accuracy : 98.4818%
   Confusion matrix:

	Predicted(Black)	Predicted(White)
Actual(Black)	24223634	192500
Actual(White)	187120	402826

4. **HighwayI:**
   Accuracy : 89.7415%
   Confusion matrix:

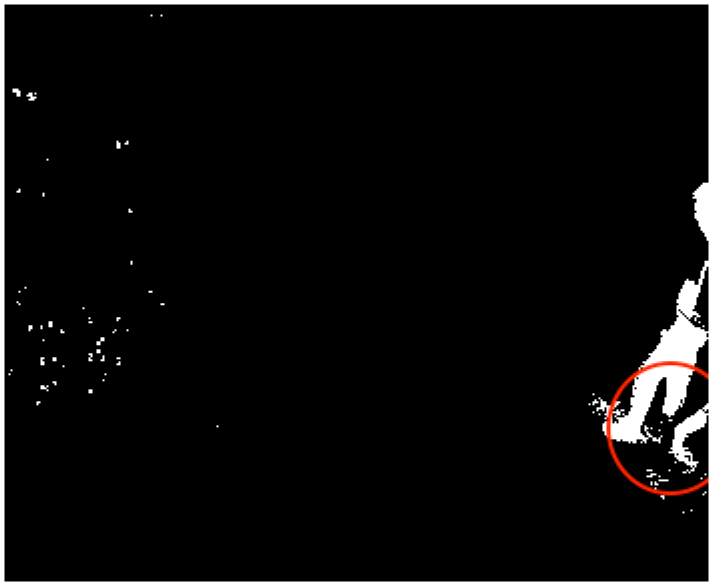
	Predicted(Black)	Predicted(White)
Actual(Black)	28973899	1557729
Actual(White)	1908806	1351566

5. **IBMtest2:**
   Accuracy : 97.9314%
   Confusion matrix:

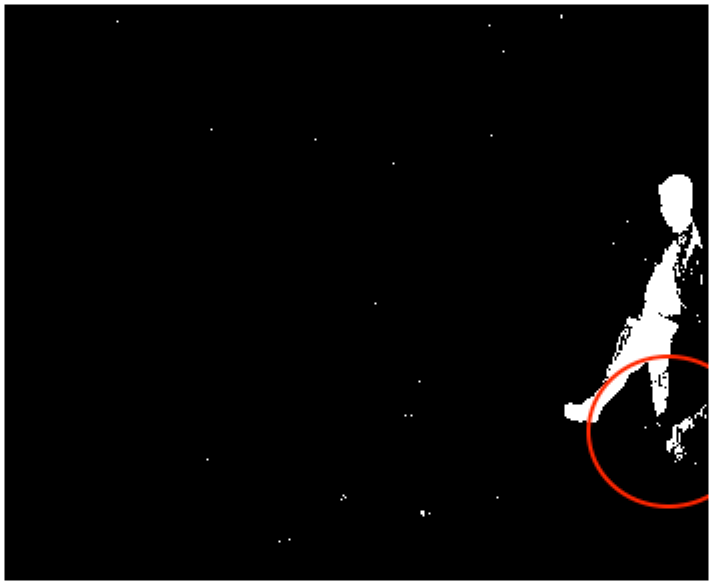
	Predicted(Black)	Predicted(White)
Actual(Black)	6562325	37391
Actual(White)	105588	206696

Key Observations:

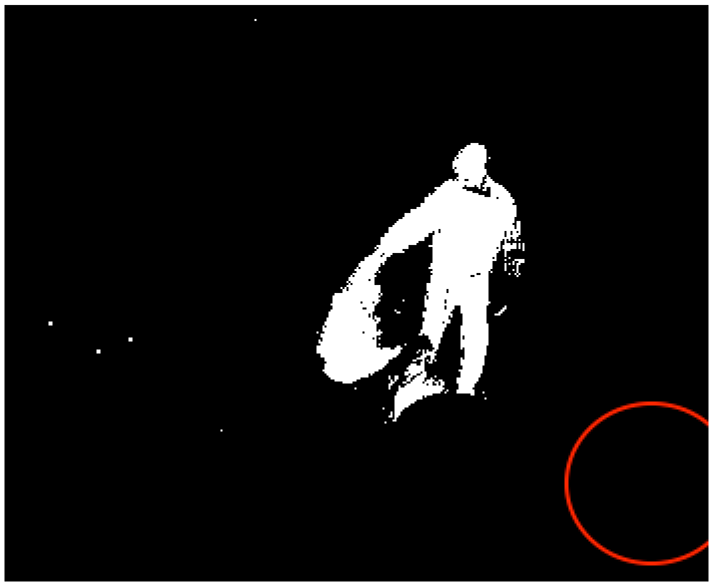
1. Since the base model is based on the first frame, if there is any object in the first frame it will be considered as background. In the subsequent frames when that object moves, i.e. according to the base model there is a change then that region is coloured white, which ideally should not have been the case. E.g. In the first dataset i.e. Candela_m1.10, the first frame has some part of the leg of the person since he is entering that frame, so throughout the other frames that area stays white. So to overcome this, I played around with the learning rate and the value of w in equation (5). Initially I was taking cnt as 50 and w as 0.1 but after playing around a little I came to the conclusion that cnt = 1 and w = 0.3 gave better results and the model slowly learnt and the false foreground detection reduced eventually in the subsequent frames.
   An example of this observation is given below:



Frame 12: As we can see there is false foreground detection encircled in red.



Frame 16: visible diminution of the false foreground detection.



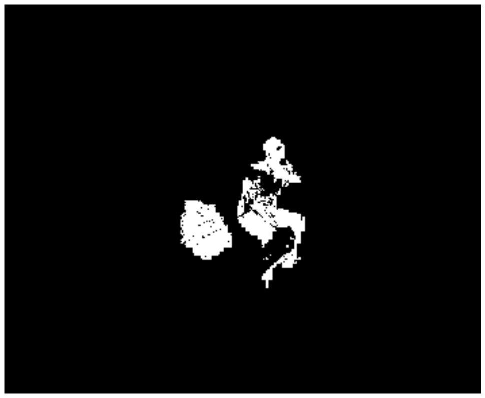
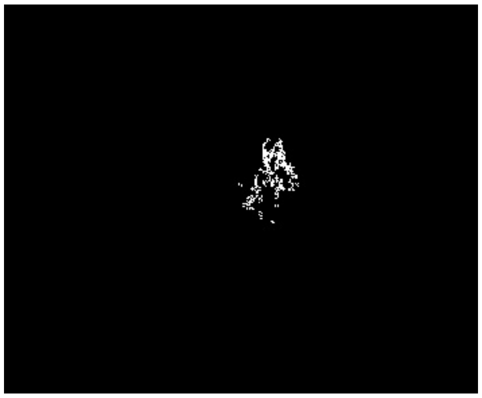
Frame 54: No more false foreground detection.

2. Another reason for false foreground detection is the illumination changes on the walls and floor due to shadows since shadows lead to change in intensity of the pixels.

<u>Comparison:</u>

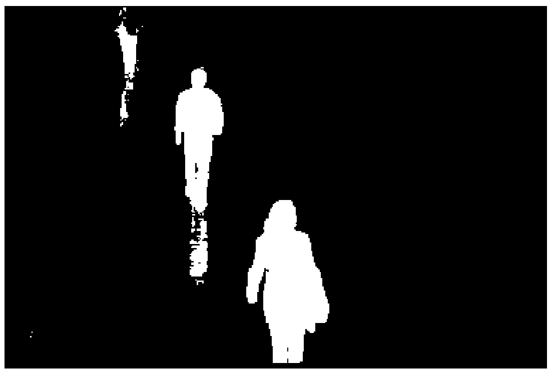
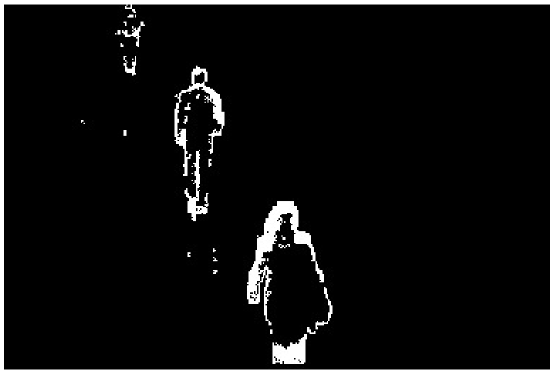
(with 2022MCS2052(Harsh Singh Chauhan)) model(Y=0) : Gaussian Mixture Model for Background Subtraction with exponentially decaying weights for the past values

1. After some time if an object is immobile in a frame for long, in GMM that object gradually diminishes as that immobile object becomes a part of the background after some time. In KDE the object fades out a little but does not completely diminish whereas in GMM it was observed that the object completely faded after some time.
   Frame 137(Candela_m1.10):

KDE GMM

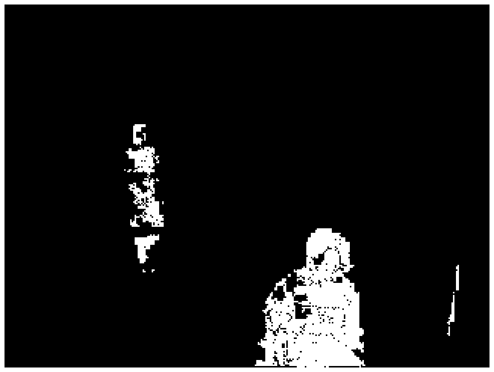
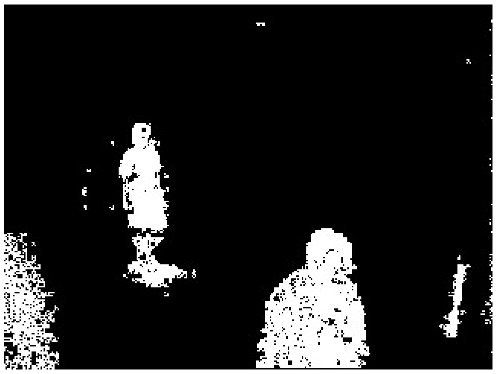
2. For GMM, in some of the input datasets after a point we could only see the outline of an object but the object was hollow from inside, whereas in KDE we could see the object wasn't hollow. A reason for this could be a continuation of the first point mentioned, since when an object is moving it does not move at God's speed so after a point the inner area of the object, although it is moving but since the color of the area is similar, therefore the intensities are similar so even when the object is moving the pixel intensities at a particular point remain same for a longer period of time, hence are considered background and the moving object is thus hollow.
   Frame 410(CAVIAR1):

KDE GMM

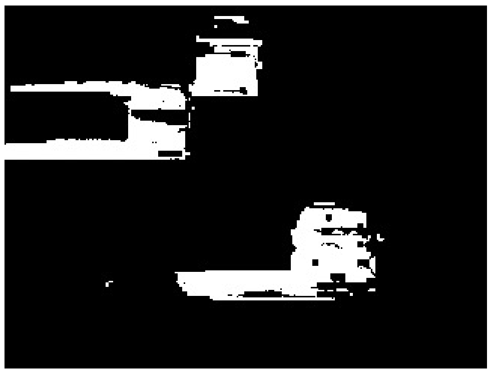
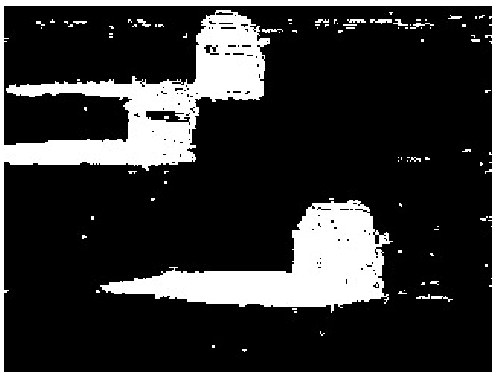
3. A few more examples to show that noise is lesser in KDE than in GMM.

Frame 47(IBMTest2)

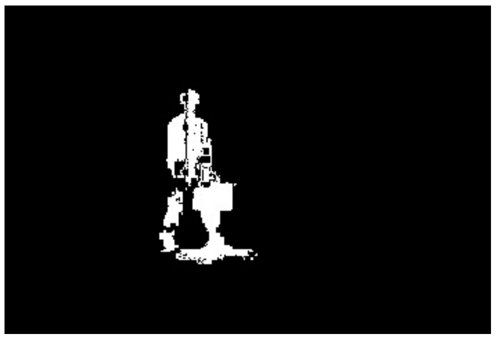
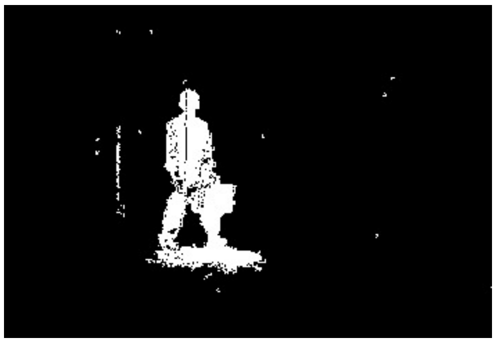
KDE GMM

Frame 120(HighwayI)

KDE GMM

Frame 42 (HallAndMonitor)

KDE GMM

4. The accuracy and confusion matrix shows that KDE did a better job in foreground detection and background subtraction since the accuracy of the KDE model is better than

the GMM model.

Dataset	KDE accuracy	GMM accuracy
Candela_m1.10	98.40%	94.27%
CAVIAR1	98.73%	94.79%
HallAndMonitor	98.48%	94.43%
HighwayI	89.74%	87.54%
IBMtest2	97.93%	85.8%

References:

1. An Adaptive Background Subtraction Method Based on Kernel Density Estimation - Jeisung Lee and Mignon Park (https://www.mdpi.com/1424-8220/12/9/12279)
2. https://theailearner.com/2018/10/15/creating-video-from-images-using-opencv-python/