Computer Vision COL780

Shashank Krishna Vempati
2022AIY7509

Introduction: (Y = 1 : Same weight for last N pixel values)

This code is written in Python and uses the OpenCV library. It is a simple implementation of a Gaussian Mixture Model (GMM) background subtraction algorithm to detect foreground objects in a video sequence. The code first reads a series of images in a folder, processes them into grayscale, and stores the processed frames in a list of N number of frames.

The GMM algorithm starts by initializing the N number of frames as the background, with each pixel of the frames represented by the K number of Gaussian distributions (where K is specified by the user). The algorithm then calculates the mean (mu) and standard deviation (sigma) of each Gaussian distribution and assigns equal weight (w) to each of the K Gaussians for each pixel.

As the video sequence progresses, the algorithm uses the next frame in the list to update the Gaussian distributions for each pixel. The algorithm matches the intensity value of each pixel in the current frame to the K Gaussian distributions. If the intensity value of a pixel falls within 2.5 standard deviations of a Gaussian distribution, the algorithm updates the mean and standard deviation of that Gaussian to reflect the current intensity value of the pixel. The weights of the Gaussian distributions are also updated based on their proximity to the current intensity value.

The algorithm then sorts the K Gaussian distributions for each pixel in decreasing order of their updated weights and discards the lowest weighted Gaussian. The process is repeated for every frame in the video sequence.

The final result of the algorithm is a binary image, where the pixels that belong to the background are set to 0, and the pixels that belong to the foreground are set to 255. The code then writes the binary images to an output video file for visualization.

The algorithm also does background noise suppression and then draws bounding boxes (solid rectangles) for the objects that are moving i.e objects that are detected as foreground.

The code also removes noise and post-processes the frames to detect connected components with desired size range and aggregates the pixels of these components. The output is stored in a new video. The post-processed frames are displayed on the screen. Additionally, there is a "merge" function that merges two rectangles into one if they overlap or are close to each other, and an "overlap" function that checks if two rectangles overlap.

Equations for updation of parameters:

*mu_t+1 = (1-rho) * avg + rho*

*sigma_t+1 = sqrt(((1-rho) * std_last_n_frames + rho * (curr_pix - mu_t+1)**2) / N)*

Where "avg" is computed from last N frames as a mean (equal weight to last n frames)

File hierarchy:

```
2022AIY7509
.
├── CAVIAR1_Output
│   ├── CAVIAR1_Foreground.mp4
│   ├── CAVIAR1_Foreground_BoundingBox.mp4
│   └── CAVIAR1_Foreground_Processed.mp4
├── Candela_m1.10_Output
│   ├── Candela_Foreground.mp4
│   ├── Candela_Foreground_BoundingBox.mp4
│   └── Candela_Foreground_Processed.mp4
├── HallAndMonitor_Output
│   ├── HallAndMonitor_Foreground.mp4
│   ├── HallAndMonitor_Foreground_BoundingBox.mp4
│   └── HallAndMonitor_Foreground_Processed.mp4
├── HighwayI_Output
│   ├── HighwayI_Foreground.mp4
│   ├── HighwayI_Foreground_BoundingBox.mp4
│   └── HighwayI_Foreground_Processed.mp4
├── IBMtest2_Output
│   ├── IBM_Foreground.mp4
│   ├── IBM_Foreground_BoundingBox.mp4
│   └── IBM_Foreground_Processed.mp4
├── main.py
└── requirements.txt
```

Running the code:

There is a “main.py” file in the folder which can be solely used to produce all the results that are stated above. Remaining codes are parts of the main code to test the output and debug.

Command to run: `python main.py ./VideoFramesFolder/input`

Need to paste the “main.py” file outside the folder containing input frames.

Requirements:

Python version: 3.8.15
Modules: OpenCV, numpy

Observation & Failure Cases:

In my implementation of the GMM based background subtraction algorithm, I utilized a fixed set of parameters across all input datasets. However, it was observed that variations in the frame rates (fps) and background dynamics (fast and slow pace variations) in different videos had a significant impact on the results produced by the algorithm. The use of constant weights for the last N pixels in the algorithm resulted in an output with excessive noise and a lack of robustness. To mitigate these issues, it is recommended to use either decaying weights for all previous frames or kernel density estimation (KDE) for a more robust solution.

The presence of a halo around moving objects in the frame is due to the averaging of the last N frames. This value of N, which I set to 5, can have an impact on the magnitude of the halo effect.

To improve the output of the algorithm, I applied post-processing techniques such as the elimination of blobs with sizes smaller than a predetermined threshold, followed by windowing (with a 5x5 kernel) and noise removal. To further enhance the efficiency of the algorithm, I utilized integral images prior to the noise removal step, reducing the computational complexity.

The GMM based background subtraction algorithm is an adaptive approach, which adjusts to slow-changing variations in the input. The algorithm identifies pixels that belong to either the background or foreground based on the variations observed at each pixel.

In my analysis of the output videos of CAVIAR1 and HighwayI, I observed a noticeable difference in their performance. The CAVIAR1 video has a higher frame rate, which allows the GMM-based background subtraction algorithm to adapt to changes in the video gradually and steadily. As a result, the output video is smoother and less prone to errors compared to HighwayI.

The HighwayI video, on the other hand, has a more dynamic nature, with rapid movement of cars and significant differences between each frame. In this scenario, the average of the last N frames creates more disturbance in the output, leading to a less accurate representation of the background. This highlights the importance of carefully considering the characteristics of the input data when implementing a background subtraction algorithm.

Additionally, even small changes in illumination, as observed in the Candela_m1.10 input, can result in frequent fluctuations in the output, which can be challenging to mitigate using a constant weight approach. In such cases, using adaptive methods like decaying weights or kernel density estimation may be more appropriate.

Finally, it should be noted that various thresholds can be used as hyperparameters in the algorithm, which can be fine-tuned to suit the specific scenario, implementation purpose, and deployment environment.

Comparison: (with Y=2 : Prafful):

One of the key disparities between Gaussian Mixture Model (GMM) based and Kernel Density Estimation (KDE) based background subtraction is the processing time. GMM models update per pixel, and as the number of Gaussians increases, the processing time also increases.

In my approach, the shape of moving objects takes time to adapt, resulting in an aura-like appearance around the images, which is not desirable. However, averaging the last n frames does intensify this effect, which is not simply the case with KDE based background subtraction.

It was observed that some videos exhibit a blinking effect due to GMM's difficulty in adapting to slight changes in lighting conditions. The GMM model with constant weights for the last n frames struggles to adapt, causing the output videos to exhibit blinking effects under light conditions.

My approach requires a significant amount of parameter tuning specific to each scenario, and its robustness in separating background and foreground is limited.

The clarity of the output is significantly improved in the KDE based approach. However, the GMM approach requires additional pre-processing steps to enhance its robustness and produce improved results.

In summary, the KDE based approach outperforms the GMM based background subtraction in terms of output quality and efficiency. These estimation techniques demonstrate the evolution of background subtraction from basic methods to advanced, highly efficient techniques.

Link to Output Videos:

OneDrive Link:
COL 780 Computer Vision Assignment 1

References:

OpenCV:
https://pyimagesearch.com/2021/02/22/opencv-connected-component-labeling-and-analysis/

Adaptive background mixture models for real-time tracking:
http://www.ai.mit.edu/projects/vsam/Publications/stauffer_cvpr98_track.pdf

Integral Images:
https://computersciencesource.wordpress.com/2010/09/03/computer-vision-the-integral-image/