COP290 - Traffic Density Estimation

Karan Aggarwal, Samanyu Mahajan

2019CS10699, 2019CS50446

1 Metrics

Error Metric: We have provided the user with two choices for the error metric:

1. **abs**: Mean of the *absolute value* of individual errors. We have used this as a default and all further analysis is done on this.
2. **rms**: Mean of the *squares* of individual errors.

Utility Metric: Lower Error implies greater Utility. We obtain a utility score, from 0-100 as follows:

$$Error = \frac{\sum_{i=1}^{N-1} d_i - d_{0i}}{N-1}$$
$$Utility = (1 - Error) \times 100$$
$$Average\ Utility = \frac{Queue\ Utility + Dynamic\ Utility}{2}$$

where N = Total number of frames in the video, d_i = Density of the i^{th} frame of the result, and $d_0 i$ = Density of the i^{th} frame of the Baseline.

Note: We have not processed the first frame as dynamic density would trivially be 0 for that.

Since we have implemented both Queue and Dynamic Densities, we obtain Queue Utility and Dynamic Utility. To obtain a single value of $Utility$, in the calculation of $Error$, we include the absolute difference in both Queue Density and Dynamic Density, compared to Baseline. This is mathematically equivalent to taking the average of Queue Utility and Dynamic Utility, and so we obtain a single utility metric namely, $Average\ Utility$.

Run-time Metric: Time taken to execute the program in seconds

We calculate the total processing time. This does not include the file input/output time. We use $std :: chrono :: high\_resolution\_clock :: now()$ which returns a time point representing the current point in time.

The start and stop time points are obtained as those before and after the execution of the method. Their difference gives us the Run-time Metric.

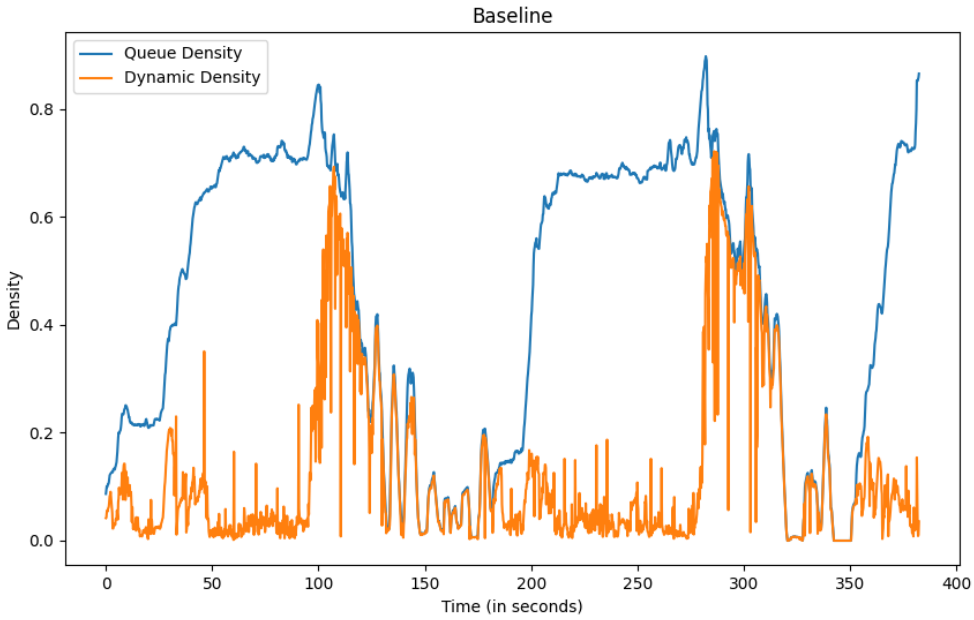
Our aim is for a high Utility and low Run-time.
For the Baseline,

$$Utility = 100.00$$
$$Run - time = 242\ seconds$$

2 Methods

The benchmark for our analysis is the video of Lajpat Nagar Junction provided: *trafficvideo.mp4*.

Baseline



Queue Density and Dynamic Density vs Time for Baseline Method

We have implemented **single-threaded dense optical flow** with full resolution and without frame-skipping as the baseline. Queue Density and Dynamic Density of each frame is saved to an output file, along with the total run-time, for further analysis.

Method 1 (Skipping Frames)

Sub-sampling frames - processing every X frame i.e process frame N and then frame $N + X$, and for all intermediate frames just use the value obtained for N.

Parameter: A positive integer X

Note: $X = 1$ is equivalent to baseline

Method 2 (Reducing Resolution)

Parameters: Two positive integers X, Y denoting the width and height in number of pixels to specify the resolution. We resize the each frame read from the video to the specified resolution before processing.

Note: Original resolution of benchmark is [1920 x 1080] so parameters: $X = 1920$, $Y = 1080$ is equivalent to baseline.

Method 3 (Spatial splitting of frames across threads)

Let X be parameter equal to the number of threads. Each frame is divided into X rectangles of equal area.

These could have been taken vertically or horizontally. For our purpose, they were **taken horizontally** as it provided greater utility. These rectangles are distributed across the threads to be processed independently.

After each thread is finished, the individual results of densities are **combined by taking the average weighted by the respective areas of the rectangles**. Note that we could not simply take the unweighted average as their might be 1 frame with a few extra pixels, if X did not divide the height of the frame.

Note: $X = 1$ is equivalent to the Baseline.

Method 4 (Temporal splitting of video across threads)

Let X be parameter equal to the number of threads. Consider N to be the total number of frames. Each thread is given $\lfloor \frac{N}{X} \rfloor$ consecutive frames of the video to process independently.

After each thread is finished, the **individual results of densities are simply appended to each other to get the final result**.

$X = 1$ is equivalent to the Baseline.

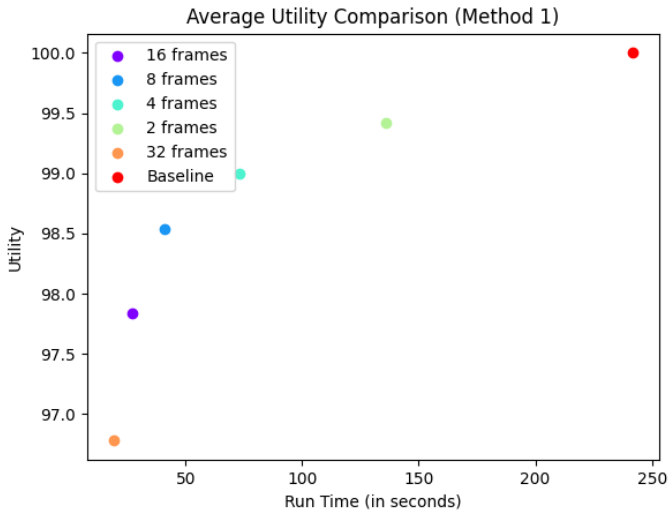
Method 5 (Sparse Optical Flow)

Sparse extracts interesting features from image and computes the motion vector for this specific set of interesting points (for example – detected corners on image). This is then used for Optical Flow calculation, assuming that the neighboring pixels have to some extent the same motion vector. Dense Optical Flow, on the other hand, calculates a motion vector for every pixel in the image.

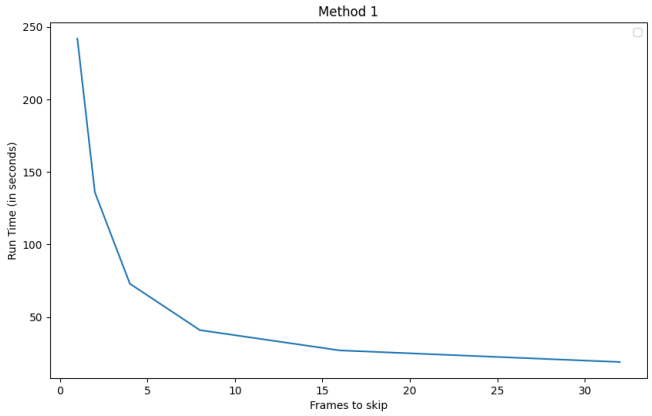
3 Trade-off Analysis

Method 1

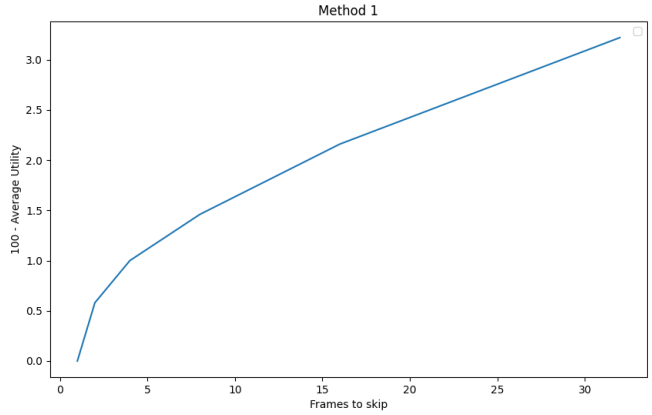
Frames to skip	Queue Utility	Dynamic Utility	Average Utility	Run Time (s)
2	99.89	98.94	99.42	136.0
4	99.70	98.30	99.00	73.0
8	99.36	97.72	98.54	41.0
16	98.72	96.96	97.84	27.0
32	97.61	95.94	96.78	19.0
Baseline (1)	100.00	100.00	100.00	242.0



Utility vs Run-time Trade-off for Method 1



Run Time vs Frames to skip (X) for Method 1. Run-time varies inversely with X



100 - Average Utility vs Frames to skip (X) for Method 1. Linear variation

As X, number of frames skipped is increased,

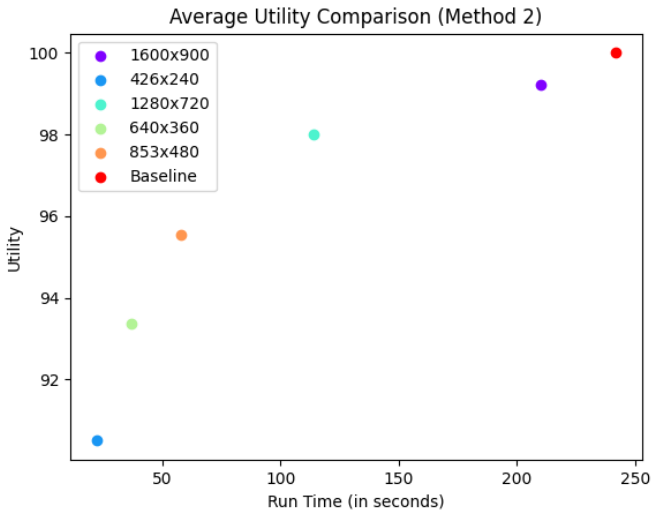
1. **Run-time Decreases:** Let $total$ = number of frames processed by baseline. We process lesser number of frames, (namely $\lfloor \frac{total}{X} \rfloor$ frames), the total processing time reduces. As most of the time is taken in frame processing, run-time almost reduces by a factor of X. From the table for method1 observe that on doubling X the run-time almost reduces by a factor of 2. Further from plot of run-time vs X we see that indeed run-time varies inversely with X.

2. **Utility Decreases:** Since we let intermediate frames (N to $N + X - 1$) values to be the same as that of frame N, these differ from the actual values in baseline. This increases the cumulative_error and decreases the utility. Note that error occurs on the frames that we skip so at an average the cumulative_error or $[100 - utility]$ is proportional to the number of frames we skip (X). Observe that $[100 - utility]$ almost doubles on doubling X. Further from $[100 - utility]$ vs X plot we see that utility varies linearly with X

In conclusion, as number of frames skipped increases, run-time and utility decrease.

Method 2

Resolution	Queue Utility	Dynamic Utility	Average Utility	Run Time (s)
426 × 240	91.32	89.69	90.50	22.0
640 × 360	93.58	93.19	93.38	37.0
853 × 480	95.89	95.21	95.55	58.0
1280 × 720	98.46	97.53	98.00	114.0
1600 × 900	99.72	98.74	99.23	210.0
Baseline (1920 × 1080)	100.00	100.00	100.00	242.0



Utility vs Run-time Trade-off for Method 2

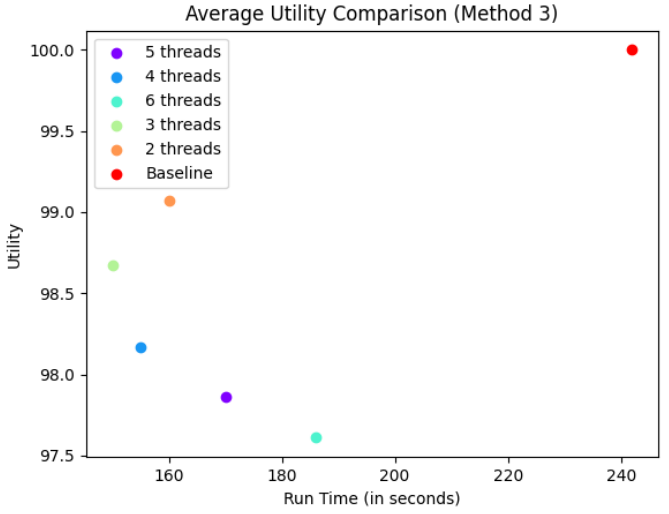
As the resolution decreases,

1. **Run-time Decreases:** lowering resolution means resizing the frame to a smaller size. images are essentially 2-D matrix of pixel intensity values. All frame processing operations (such as warping perspective, background subtraction and calculation of optical flow) take time proportional to the size of the matrix. so run-time decreases
2. **Utility Decreases:** In the resize operation we lose precision which increases cumulative_error and decreases utility.

We conclude that lower resolution frames are processed faster, but have low utility.

Method 3

Number of Threads	Queue Utility	Dynamic Utility	Average Utility	Run Time (s)	%CPU
2	99.97	98.17	99.07	160.0	415
3	99.96	97.38	98.67	150.0	540
4	99.86	96.48	98.17	155.0	620
5	99.79	95.94	97.86	170.0	700
6	99.71	95.51	97.61	186.0	750
Baseline (1)	100.00	100.00	100.00	242.0	165



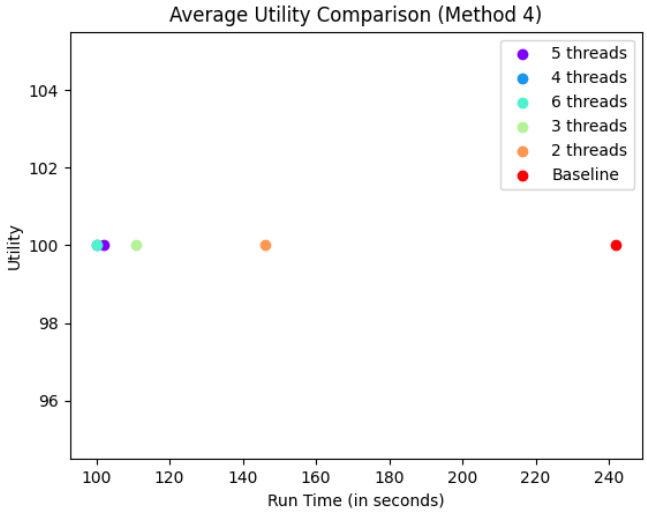
Utility vs Run-time Trade-off for Method 3

Increasing the number of threads (X) decreases the utility, as there would be some errors, especially in the case of Dynamic Density, where flow between 2 adjacent rectangles is not captured at all.

Also, observe that the run-time decreases till $X = 3$, and increases with further increasing X. This is because multi-threading speed is limited by the number of cores in the CPU. Increasing X further comes with the overhead of thread switching, which exceeds the benefit of having more parallelization for $X \geq 4$, thus increasing run-time.

Method 4

Number of Threads	Queue Utility	Dynamic Utility	Average Utility	Run Time (s)	%CPU
2	100.0	100.0	100.0	146.0	330
3	100.0	100.0	100.0	111.0	470
4	100.0	100.0	100.0	100.0	580
5	100.0	100.0	100.0	102.0	640
6	100.0	100.0	100.0	100.0	710
Baseline (1)	100.0	100.0	100.0	242.0	165



Utility vs Run-time Trade-off for Method 4

The utility remains constant at 100.00 for any X number of threads. This is because while splitting the video temporally, there is 0 margin of error as the task being done is the same as baseline, it is just split into multiple threads.

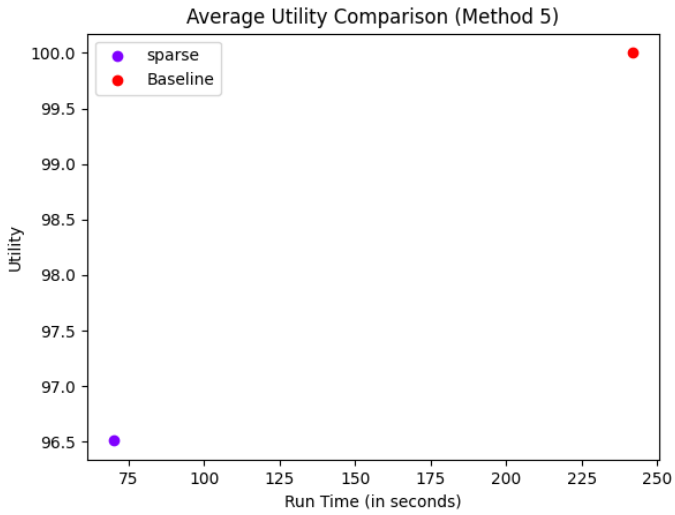
Note that we could not achieve full utility for any of the above methods as each of them introduce at least some difference with the baseline.

Observe that the run-time saturates approximately at $X = 4$. Further increasing X has no impact on the run-time because of the increased overhead of thread switching caused by the limited cores in the CPU.

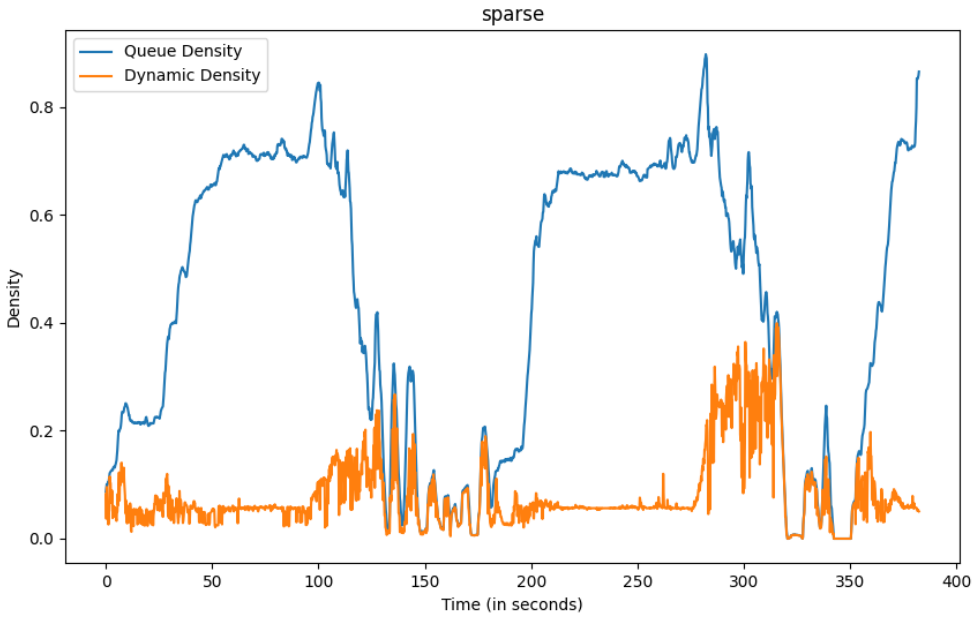
We have a clear winner at $\mathbf{X = 6}$, which has the highest utility and lowest run-time. However, 6 threads require more energy and might not be suitable for a roadside embedded board.

Method 5

Optical Flow Type	Queue Utility	Dynamic Utility	Average Utility	Run Time (s)
Sparse	100.0	93.02	96.51	70.0
Baseline (Dense)	100.0	100.00	100.00	242.0



Utility vs Run-time Trade-off for Sparse Optical Flow



Queue Density and Dynamic Density vs Time for Sparse Optical Flow. Compare with Baseline and notice the extremely high Standard Deviation.

1. **Run-time:** 70.0 compared to 242.0 seconds for baseline(dense)
2. **Utility:** Dynamic Utility = 93.02. Since the change is only in optical flow method the queue density remains same and consequently Queue Utility is 100.00.

Using only a sparse feature set means that we will not have the motion information about pixels that are not contained in it. This leads increase in cumulative_error and therefore a lower utility. As seen from the graph, sparse follows similar overall trends compared to baseline, for instance, the dynamic density drops around red light and increases around green light. However, note that for sparse, since only a small number of points are considered, the values of dynamic density obtained are smaller than those for dense. There is consequently a high standard deviation, compared to the baseline.

In conclusion, Sparse is faster, but has errors compared to dense.