Traffic Estimation Report

Arka Mandal

25 March 2021

1 Introduction

Earlier we had written a code to calculate optical flow of the video given to us. We are now supposed to write the same code in 5 different methods and check if they help in getting better results or faster results. The first thing we did was write all the 5 methods and checked if it was working using a cout. Now we need to come up with a method to check for accuracy/similarity of result with original code and a method to calculate efficiency of computation.

2 Analysis Strategy

Accuracy: Firstly, we need to get a csv file which consists of frame number, moving density and queue density in the same order in all files. We then use this csv file and convert moving density and queue density to their corresponding values between 0 and 1 where 0 is the lowest value and 1 is highest one. Python scripts used to do this is provided. We calculate accuracy by taking the square of the difference between values provided by the baseline and the code being tested. We use this error to calculate root mean square of the entire benchmark.

Efficiency: Efficiency has been calculated by two ways. Firstly the most important parameter, time: the time required to process one frame. This is simple to calculate. We run the code for some time and calculate the time taken by it. We divide it by total frames. The second parameter is load on the system. We used CPU to calculate the data. We monitored the CPU usage at fixed intervals. The mean of all these values is decided to be the CPU Load for the code. We could not come up with an efficient way to calculate CPU temperature. There are 3rd party softwares available but temperature also varies bases on when the code is run, environment etc so this parameter has been dropped. We tried varying parameters to analyze best value for each method too. Only the best has been used for analysis among methods.

CPU Load has been calculated by taking screenshots of System Monitor. Average of all CPU Loads is taken in all screenshots and average of all screenshots is taken to get CPU Load for the execution. An example of screenshot is shown below.

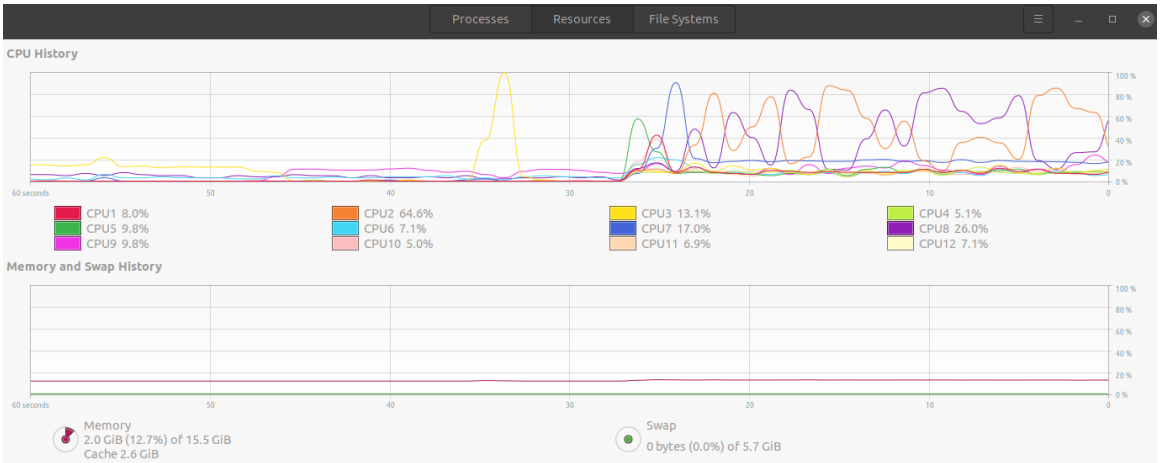

Figure 1: System Monitor

3 Important Notes before Analysis

- Method1 to 5 with parameter n are referred to as FrameSkipn, LowResn, Sparse, FrameSplit(V/H)n(for vertical and horizontal splits respectively), VidSplitn respectively. These notations are used both in file naming conventions and this document.
- Possible errors/bugs are reported at the end of the document.
- The Parameters used by us are: 2,3,4,5,6 for FrameSkip. For threading methods, we use 1-6. For LowRes, we use 0.75,0.5,0.33,0.25,0.15. For BaseLine and Sparse methods there is no parameter used by us. These are referred to as parameter index, 0 to 5 exactly in this order.
- Some graphs are combined in order to provide a big picture in small space.
- Some graphs done by us are not particularly important here so they are provided separately in a folder called 'Plots'.
- All plots where y axis is 0 to 100 is a merged normalized plot. Values are normalized within each level. Comparison between different plots should not be done.
- Most normalization are done to make plots clear. Small changes in values are difficult to understand. Normalized graphs can be easily visualized and also compared with others.
- All executions are done under as similar conditions as possible: all apps closed, internet off, charge full etc. Deviations however may occur due to changes in temperature.
- All generated csv files and plots have been provided along with some txt files where data is collected manually concisely. Python scripts have also been provided that are used by us. Some scripts have been used multiple times for various purposes after slight modifications.
- BaseLine is the same code as subtask(b).

4 Approach of Methods

4.1 Method 1:Skipping Frames

We process every n^{th} frame where n is passed as parameter. We assume that the changes in density in subsequent frames are very less and thus we keep same values as the x^{th} frame for the next n^{th} frames. The more the value of n, the faster our code will be as we skip more frames and avoid optical flow calculation but accuracy will drop as we ignore changes in consecutive frames. A graph is shown where we show accuracy vs n and efficiency vs n.

4.2 Method 2:Sparse Flow

Our original code used dense flow. We used sparse flow here. There are no parameters to compare here as there are no parameters. We have used standard values for ShiTomasi algorithm just as we have used standard values in the Farneback Optical flow algorithm.

4.3 Method 3:Resolution Down-sampling

Here we take in a double from the user and reduce both height and width of the video by the same value. For example, input of 0.5 reduces rows and columns of the image matrix by half. We check this for a few values of parameter. Let this value be n. We plot a graph of accuracy vs n and efficiency vs n. The plotted graphs are shown below.

4.4 Method 4:Splitting Frames

Here we take number of threads as input and we split frames by that amount. Each part is given to one thread to calculate. After one frame is processed all data is calculated for that frame. Accuracy and efficiency plots are given in a later subsection. This is again subdivided into two parts: vertical and horizontal splits of frames

4.5 Method 5:Trimming Video

Here we divide the entire video file into as many sections as number of threads and give each part to each thread. After the csv is made, we need to sort entire data for further analysis.

5 Analysis in Parts

5.1 Plots for Accuracy

The plot below shows accuracy of moving and queue densities separately. The labels are not to be compared with each other. They only provide comparison among its other parameters with the lowest error as 100 and highest as 0. From here, we see the plot where the value of accuracy is highest other than where parameter value is 1.

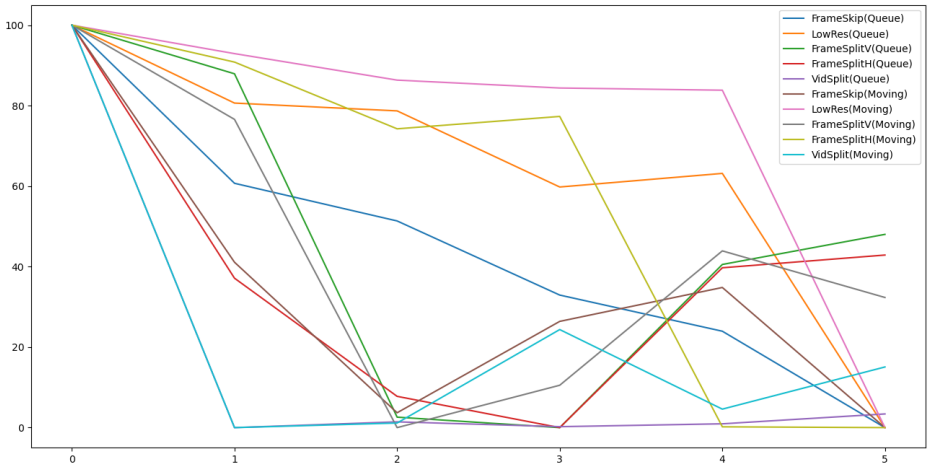

Figure 2: Accuracy%(Normalized) vs Parameter Index

We add the moving and queue density accuracy and we call it our net accuracy. We see where we get the best net accuracy parameter wise for each method. A plot is given below where net accuracy is plotted. Also, y axis is relative to the label. Different methods are not to be compared with each other. Only comparison within a method between different parameters is done here and plotted between 0 to 100.

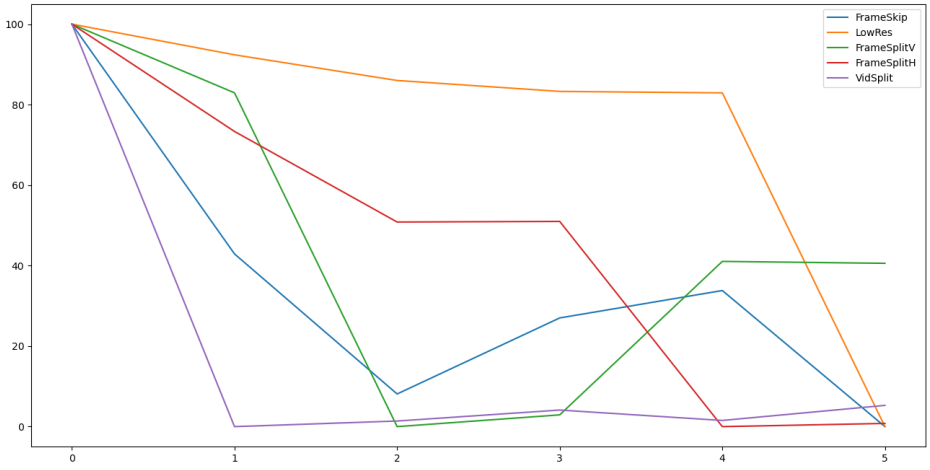

Figure 3: Net-Accuracy%(Normalized) vs Parameter Index

Here we take the peaks and this is the parameter index where best accuracy is received. We again plot this for inter-method plot but this time as sum of error vs parameter index. The index with lowest error is the best result in this case. Here we take peaks of different methods separately and that gives us the best parameter in that method domain. We use this best method for Inter-Method Analysis.

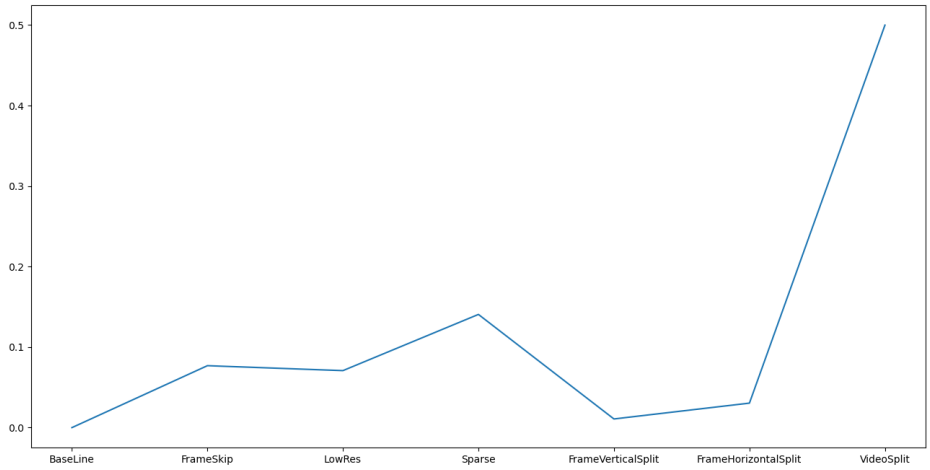

Figure 4: Inter-Method Error(Absolute)

5.2 Observations in Accuracy

- Splitting frames horizontally or vertically has similar results. Same is the case with FrameSkip and LowRes.
- For two of three thread applied methods, best result is received for number of threads=2. This is because of reduced corner cases. Splits between frames or parts of video is difficult to merge with threading which calculate independently.
- For VidSplit, Accuracy drops very fast and stays there. This is because few frames are removed from calculation due to independent computation.
- Accuracy-wise, VidSplit is the worst method and Splitting frames vertically yields best result.

- Highest accuracy in VidSplit is given for 6 threads by a small amount. This may be an anomaly. Need to get data from more samples for better analysis.

5.3 Plots for efficiency

For efficiency, 3 parameters are calculated by us: CPU load, Memory Load, Time taken to complete execution. All calculated data is uploaded in a file called 'Efficiency Stats'. We observe that there is no significant changes in Memory Load so we will ignore that for the rest of our Analysis. CPU Usage is calculated by taking screenshots of system monitor after certain intervals of time. Average of all 12 CPUs and all screenshots are taken by us manually and we call it avg CPU load.

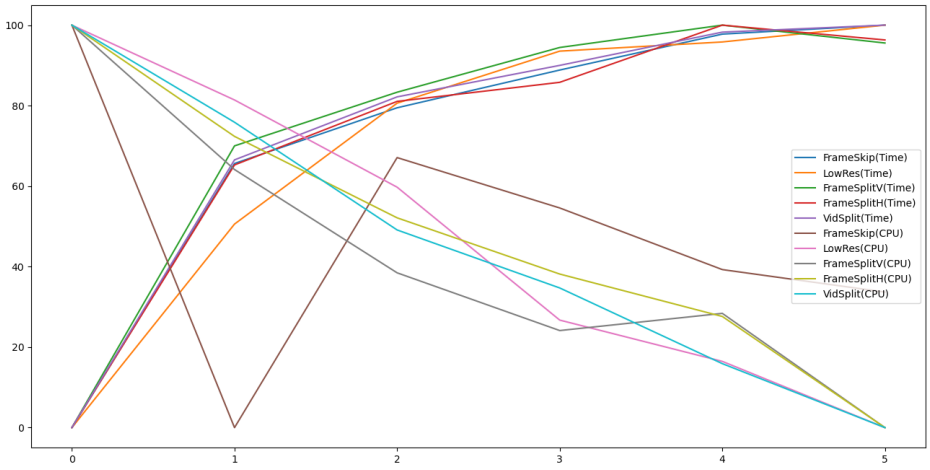

Figure 5: CPU Load% and Time Elapsed vs Parameter Index

Here we easily see that, in general, CPU Load increases and Time decreases. For most methods, CPU Load reaches saturation faster than Time Taken. This is expected from theory. From this above plot, we find the lowest times taken by each method and plot them separately for better visualization. For this plot below, y axis is in seconds. The parameters used for indices in x are 6,0.15,5,5,6 for FrameSkip, LowRes, FrameSplitVertical, FrameSplitHorizontal and VidSplit respectively.

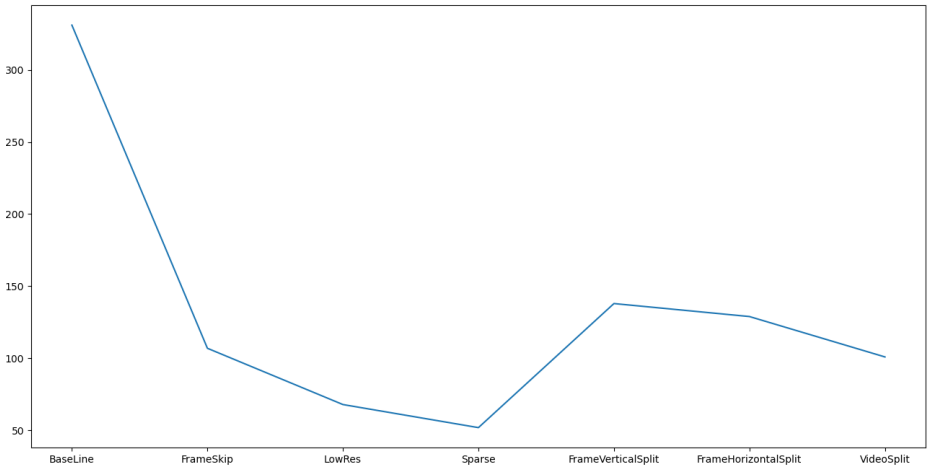

Figure 6: Time Elapsed vs Parameter Index

5.4 Observations in Efficiency

- In the first plot there is a serious deviation for FrameSkip with parameter 2. It is assumed that this happens because skipping 1 frame is not good enough. Extra calculation is done whether to skip or not and that is not coped up with the actual skip.
- Sparse is the fastest method.
- Both FrameSplit threading methods have very similar performance.
- Every single method is a serious upgradation from Base-Line. Elapsed time is less than halved in every method.

6 Inter-Method Analysis

The best results of each method has been taken and we compare accuracy and efficiency. One way to compare is by getting these values on a scale of 0 to 1. After that, we check which method has the greatest accuracy and efficiency value. One thing to note here is that, we weigh accuracy and efficiency equally here. If they were different(based on where this code would be used), we might have to take a weighted sum. Lack of information about real-life requirements lead us to weight them equally. Our decisions in real life may be influenced bu hardware, budget, energy required, type of road etc. The values of error, efficiency is shown below.

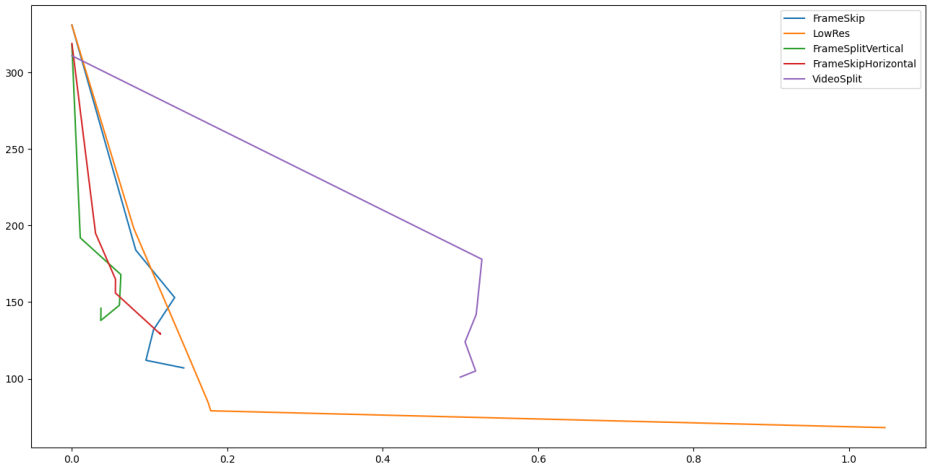

Figure 7: Time Elapsed vs Error

Here, y axis is time taken in seconds, x axis is error given by a method. This is not a normalized graph and can be directly compared. We need lesser time(lower in y axis) and lesser error(leftward on x axis). So, the points which are closer to origin refer to a better method overall. We also notice that for threading methods, increasing parameter values leads to quick decrease in time but not a serious reduction in accuracy. LowRes shows some quick results initially while increasing parameters but shows a quick decline in accuracy soon after. VidSplit shows opposite result. Accuracy drops fast first and then time taken drops. FrameSkip and FrameSplits maintains consistency throughout even after increasing parameters.

LowRes2, FrameSkip4, FrameSplitVertical5 yeilds very similar results. Anyone can be used based on time or accuracy constraints. If CPULoad is taken into consideration, LowRes2 will be the best option for this benchmark.

7 Final Comparison between BaseLine and best method analytically

We do a final density plot to check how similar the plot of Baseline and LowRes2 is.

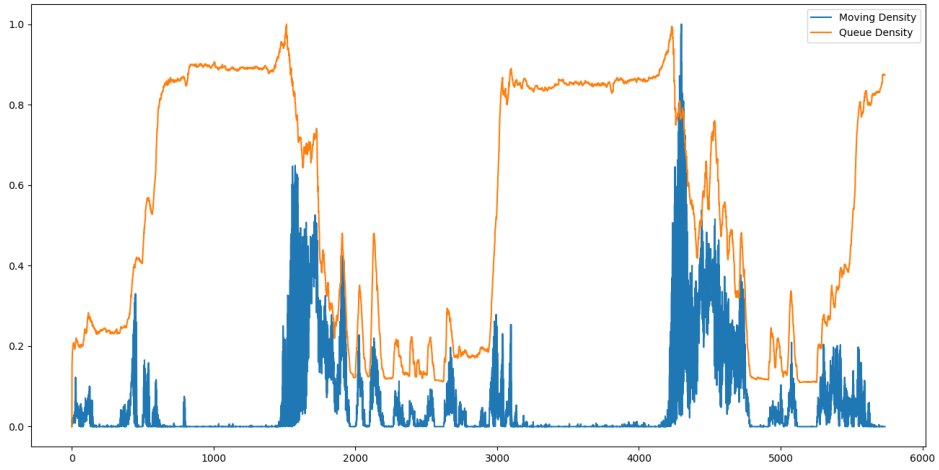

Figure 8: BaseLine Density Plot

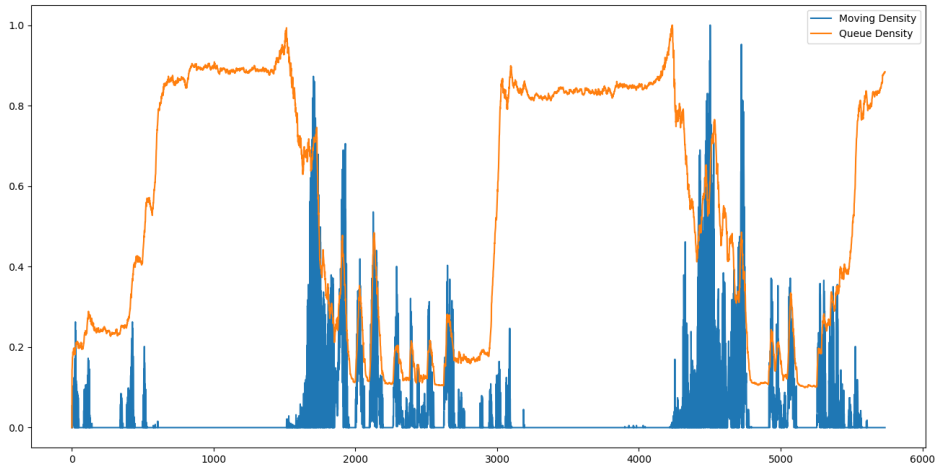

Figure 9: LowRes2 Density Plot

Even visually, the results are pretty similar. This has been achieved in $\frac{1}{3^{rd}}$ of the time compared to that of BaseLine. CPULoad increased by merely 8%.

8 Conclusion

From all the data above, we can say that there is no best method. Every method has their own set of advantages and disadvantages but for the benchmark provided, lowering the resolution by half seems to be the best method. This is partly because the original video is too detailed(1080p). There are a lot of extra information which is practically useless but need to be handled every frame. Depending on different benchmarks, the outputs may vary significantly. A better analysis is possible by taking a huge number of sample inputs and outputs.

9 Errors/Bugs and Extras

- All screenshots of system monitors for all execution is not provided. The total size is well over 300MB. It may be shown during demo to our instructors.

- Before the session on Piazza, threading methods were done using <thread.h> on windows. Those extra codes are also provided. These however have not been used in the generation of this report.
- Sparse Density Calculation could not be done by us properly. We tried the sample code in official documentation but it crashes after certain amount of frames. We got the data for Sparse by continuing calculation even after crash multiple times and combined all the data.