The Bengaluru Mobility Challenge

Guidelines for the final submission of Phase 1

Evaluation Criteria

As explained in the BTP hackathon description document, the final submissions will be evaluated on the basis of four criteria. The criteria and the corresponding weighting factors are listed below:

1. Accuracy of the predictions or estimates in comparison with ground truth (60%),
2. Novelty of the approach (20%),
3. The hardware and computational requirements (10%), and
4. The final presentation and demo (10%)

More details on how the scoring will be done for each criterion are described below.

Criterion 1: Accuracy score

Phase one consists of two problems, where each problem has two subproblems:

Problem 1

(a) The **cumulative observed number of vehicles by vehicle class** from the start of the clip to the end of the clip in a given 30-minute clip (split into two 15-minute clips) for a given camera.
(b) The **predicted cumulative number of vehicles by vehicle class** from the end time of the given clip up to 30 minutes into the future.

Problem 2

(a) The **predicted number of vehicles by vehicle class** at a previously unseen location from the starting time to the ending time of the 30-minute clip provided.
(b) The **predicted cumulative number of vehicles by vehicle class** at a previously unseen location from the end time of the 30-minute clips provided up to 30 minutes into the future.

Thus, there are four subproblems. For each of the subproblems, the submitted code will be run on 30-minute video clips, and separate scores will be computed based on the following formulas:

$$d(t,v) = \begin{cases} \min\left(|GT_{t,v} - SC_{t,v}|/GT_{t,v}, 1\right) & \text{if } GT_{t,v} > 0 \\ 1 & \text{if } GT_{t,v} = 0, SC_{t,v} > 0 \\ 0 & \text{if } GT_{t,v} = 0, SC_{t,v} = 0 \end{cases}$$

$$deviation(t,v) = \frac{1}{|T||V|} \sum_{t \in T} \sum_{v \in V} d(t,v) \times 100\%$$

T is the set of all turning patterns;
V is the set of all vehicle types;
$GT_{t,v}$ is the ground truth vehicle count for turning pattern *t* and vehicle type *v;* and
$SC_{t,v}$ is, for part (a), the vehicle count put out by the submitted code for turning pattern *t* and vehicle type *v* for the given 30-minute clip, and for part (b) the vehicle count predicted for the future 30 minutes by the algorithm for turning pattern *t* and vehicle type *v*.

The submitted code will be tested on three video clips in the case of the two subproblems of Problem 1 and the average score will be taken for each subproblem. Similarly, the code will be tested on two video clips in the case of the two subproblems of Problem 2, and the average score will be taken for each subproblem. The above process will give us one score for each of the four subproblems. The final score will be the average of the **four scores**.

Criterion 2: Novelty

The novelty of the approach will be determined by the panel of judges based on the submitted code, documentation, and the report (see section on "Items to be submitted"). Therefore, it is imperative that all novelty aspects are properly explained and highlighted in the report that is part of the final submission. Novelty could be in the technical approach, in the algorithms chosen, in the training process, in reducing the computational complexity, etc.

Criterion 3: Hardware and computational requirements

For a fair comparison, all entries will be run on a workstation with the following specifications:

CPU - Core i9
GPU - RTX4090
RAM - 64GB
SATA - 500GB

The average execution time for your submissions over all given clips will be taken into consideration to determine the score for this criterion.

Criterion 4: Final presentation

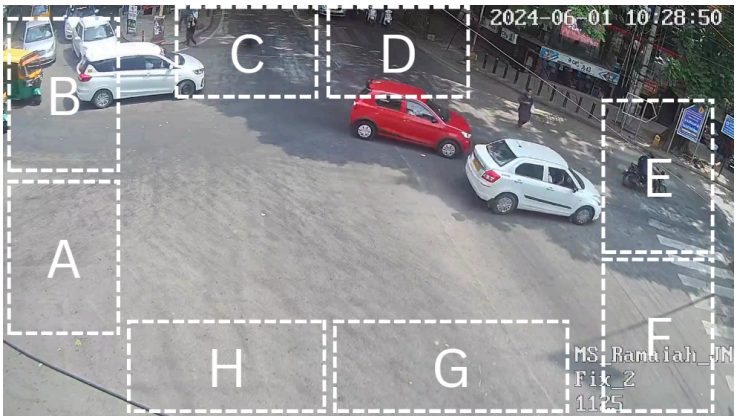
Each team will be given 15 minutes for the final presentation, including questions. The guidelines for the presentation will be shared separately.

Input & Output requirements for Docker Image

Problem 1: Here, random 30-minute video clips from any of the 20 cameras will be selected for scoring (see document called "Camera views and turning regions," which can be found on the competition website). The final score will be the average of the scores for the three video clips. Therefore, the code needs to work for all 20 cameras out of the 23 cameras for which the data has been provided to you. (We have determined that the remaining three cameras are not suitable for this task, and hence are being omitted.) You will not know the

camera ID from which the given video clip is being fetched for scoring purposes. Therefore, you need to design your code so that it works for all the given cameras.

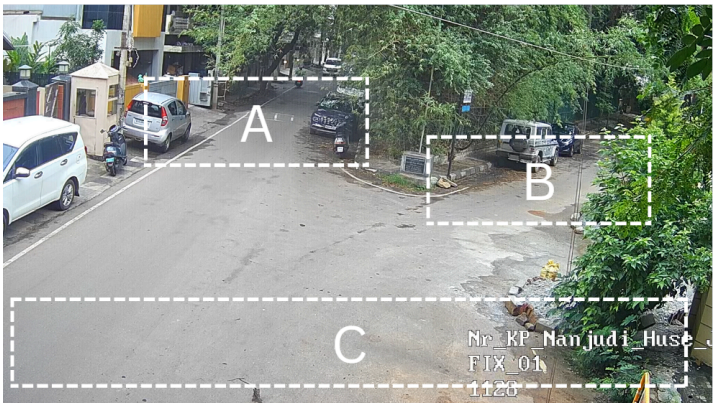
The document “Camera views and turning regions” contains (i) the camera ID, (ii) a view showing the indicative rectangular regions that can be used to define the turning patterns, and (iii) the sequence of turning patterns to be considered, for 20 cameras. An example of information for Camera ID MS_Ramaiah_JN_FIX_2 is shown below.



In the above view, there are 8 roads and a total of 12 possible turning patterns. The turning patterns are B to C, B to E, B to G, D to A, D to E, D to G, F to A, F to C, F to G, H to A, H to C, H to E. The first letter represents the origin and the 2nd letter represents the destination. These will be abbreviated as BC, BE, BG, DA, DE, DG, FA, FC, FG, HA, HC and HE. For example, BC denotes a pattern where a vehicle starts in Region B and ends in Region C. ***Note that the rectangular regions shown in the figure are illustrative***.

Problem 2: Random 30-minute video clips from any of the six “unseen” cameras will be selected for scoring (see document called “Unseen camera views and turning regions,” which can be found on the competition website). Therefore, the code needs to work for all six “unseen” cameras. You will not know the camera ID from which the given video clip is being fetched for scoring purposes. Therefore, you need to design your code so that it works for all unseen cameras.

The document “Unseen camera views and turning regions” contains (i) the camera ID, (ii) a view showing the indicative rectangular regions that can be used to define the turning patterns, and (iii) the sequence of turning patterns to be considered, for 6 cameras. An example of information for Camera ID Nanjudi_House is shown below.



In the above view, there are 3 roads and a total of 6 possible turning patterns. The turning patterns are A to B, A to C, B to A, B to C, C to A and C to B. The first letter represents the origin and the second letter represents the destination. These will be abbreviated as AB, AC, BA, BC, CA and CB. For example, BC denotes a pattern where a vehicle starts in Region B and ends in Region C. ***Note that the rectangular regions shown in the figure are illustrative.***

Based on the above reasoning, we require teams to package their code into a single Docker image such that it can be used to run on any video clip from the specified cameras for both Problem 1 and Problem 2.

The docker image needs to be formulated with the CMD section accepting a JSON file as input. It should output a JSON file named Counts.json.

The input JSON file will take this form:
{'Cam_ID': {'Vid_1': '/path_to_vid_1', 'Vid_2': '/path_to_vid_2'}}

Cam_ID represents the camera being selected for inference. Vid_1 and Vid_2 contain the path to two continuous video segments from the selected camera, which will be used for assessment. Your code needs to process both of these videos and return the observed cumulative counts for each vehicle class and each turning pattern for the given camera. In addition, it should give the same information for the number of vehicles from the end time of the clip up to 30 minutes into the future.

Both of these values should be compiled and returned in this output format:
{'Cam_ID': {'Cumulative Counts': counts_by_class_turning_pattern, 'Predicted Counts': counts_by_class_turning_pattern }}

The Cumulative Counts will be for sub-problem (a) and Predicted Counts for sub-problem (b).
counts_by_class_turning_pattern is of the format:
{Turning_Pattern: {'Bicycle': 0, 'Bus': 0, 'Car': 0, 'LCV': 0, 'Three Wheeler': 0, 'Truck': 0, 'Two Wheeler': 0}}
.
.
}

Where **Turning_Pattern** is one of the possible turning patterns for the given video. **counts_by_class_turning_pattern** will contain all possible turning patterns and the vehicle classes within it. The code should only output values for those Turning Patterns which are possible for the given camera view, rest should remain unchanged. Only the possible Turning Patterns for the respective Cam_ID will be considered for evaluation.

We have attached a sample Input & Output file for ease of understanding & use:
Input File:
https://drive.google.com/file/d/19EMxIqMXlRSYdd1uJ_d_U2fgIN4aMy7C/view?usp=sharing

Output File:
https://drive.google.com/file/d/19HL-Anae3SjFIwVypeJ-372L7teczzdd/view?usp=drive_link

The Command which will be run for assessment is:
docker run --rm --runtime=nvidia --gpus all -v <host-files-path>:<container-files-path> <image-name>:<image-tag> python3 app.py input_file.json output_file.json

Items to be filled via Google Form

A. Team name

Fill in your team name. Ensure it is the same as the one submitted during registration. Any team names not in the original Google Form responses will not be evaluated.

B. Docker Image Tag

Here you will need to paste the link of your final docker image tag of the form: ***Username/Docker Image Name:Tag Name that you would like to submit for evaluation.*** Further updation of the docker images after submissions will not be entertained, so make sure your best solution has been submitted.

C. Google Drive link of your code and report

Here, provide a sharable link of a Google Drive folder named after your team with read permissions granted to hackathon.cdpg@fsid-iisc.in and analytics@cdpg.org.in. Some of the **mandatory** components to be present in this folder are indicated below:

1. **Code and notebooks**

- Break your submission into numerous notebooks (or scripts) serving one large functionality. For example, have a notebook just to perform preprocessing, one notebook just for training, one notebook for evaluation, etc.
- Make the code modular, have neat and understandable function names and utilise object-oriented abstractions to make the code readable and modular.
- Leave sufficient comments clearly explaining the functionality of these modular blocks.
- Clearly indicate the inputs and outputs and demarcate an area of code for handling inputs and outputs
- Use a fixed seed wherever necessary to make the results reproducible.

2. Created/stored/used models

- All components which are using neural networks or machine learning must have relevant models (for e.g., Yolo, SVM, Torch, etc.), code and explanation for the same must be provided.
- Any model used must comply with the supported model formats for the libraries used, e.g., TensorFlow uses pickled or .h5 format, Torch uses .pt format, Yolo uses .pt or .onnx, etc.

3. A README.md file

- Detailed and step by step instructions to build the docker image and to execute it: Overview of Docker Build | Docker Docs
- Names and descriptions of the various notebooks and/or scripts used
- Requirements.txt describing all the libraries used in the code and corresponding versions
- Clear references to open-source models used, for e.g, if a huggingface model is used
- System requirements to run your code such as required GPU cores, RAM, CPU, etc. This needs to be consistent with the specs of the workstation that will be used for evaluation. Please see the section entitled “Evaluation.”

4. A team_name_report.pdf document containing details such as the members of the team, the approach used, etc. This report can have a maximum of 4 pages, excluding references.

- Details of the members of the team
- Introduction to the problem statement (summarising your understanding of the problem).
- Methodology - showing a clear solution architecture block diagram, describing the solution in detail.
- Results - clearly showing the performance of the solution (e.g., charts of showing the learning metrics over time, etc.) as well as prediction results on validation and test datasets.
- Conclusions, including any shortcomings of the proposed solution
- References - Must cite any relevant literature that was used in your approach, and also software used, etc.

The above Drive folder must **not** contain -

1. Input videos in the zip.
2. Modified names or folder structure of the input videos.
3. Processed video inputs (for e.g., compressed videos, etc.). If preprocessing was done on the videos, then a notebook containing the exact procedure must be provided.

Please submit the above to this Google form link: https://forms.gle/HZCqptqDVG8wX58T8

Multiple submissions from the same team will not be entertained and the form will close on the 26th of August, 2024, at 12AM IST.