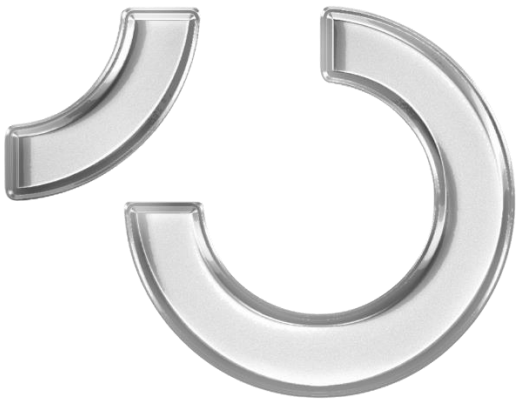
TRAFFIC COUNT AND CLASSIFICATION APPLICATION

A project to fulfil the requirement of

BALAJI INFRATECH

Proposed by

XERO TECHNOLOGIES



ABSTRACT

Traffic Count and Classification Application is an application to be offered by us in support of Classifying and tracking flow across multiple classes of Transport vehicles over visual data.

Classification, Tracking, and Counting of Vehicles is a tedious task requiring high accuracy and high attention span, and continuous monitoring of visual data. Human labour employed at the specific Task, may tend to misinterpret data, produce incorrect output, and take large amount of work time to process and complete the task.

The Traffic Count and Classification Application will be an offline application that supports the tracking, classification, and counting of vehicle classes on visual data derived from Indian Roadways. This will help the high processing needs, higher classification and counting accuracy, decrease labour cost, and provide data in a systemized manner all at a low time usage.

The application development project is divided into multiple phases / stages, each covering diverse needs and specifications for the application.

The project is covered under a commercial contract with a Commercial Survey Organization (Name un-disclosed).

INDEX

DATA SET CREATION

We have curated a Custom Dataset of 17 Indian Vehicle Classes containing Images from both Day Time and Night Time Images.

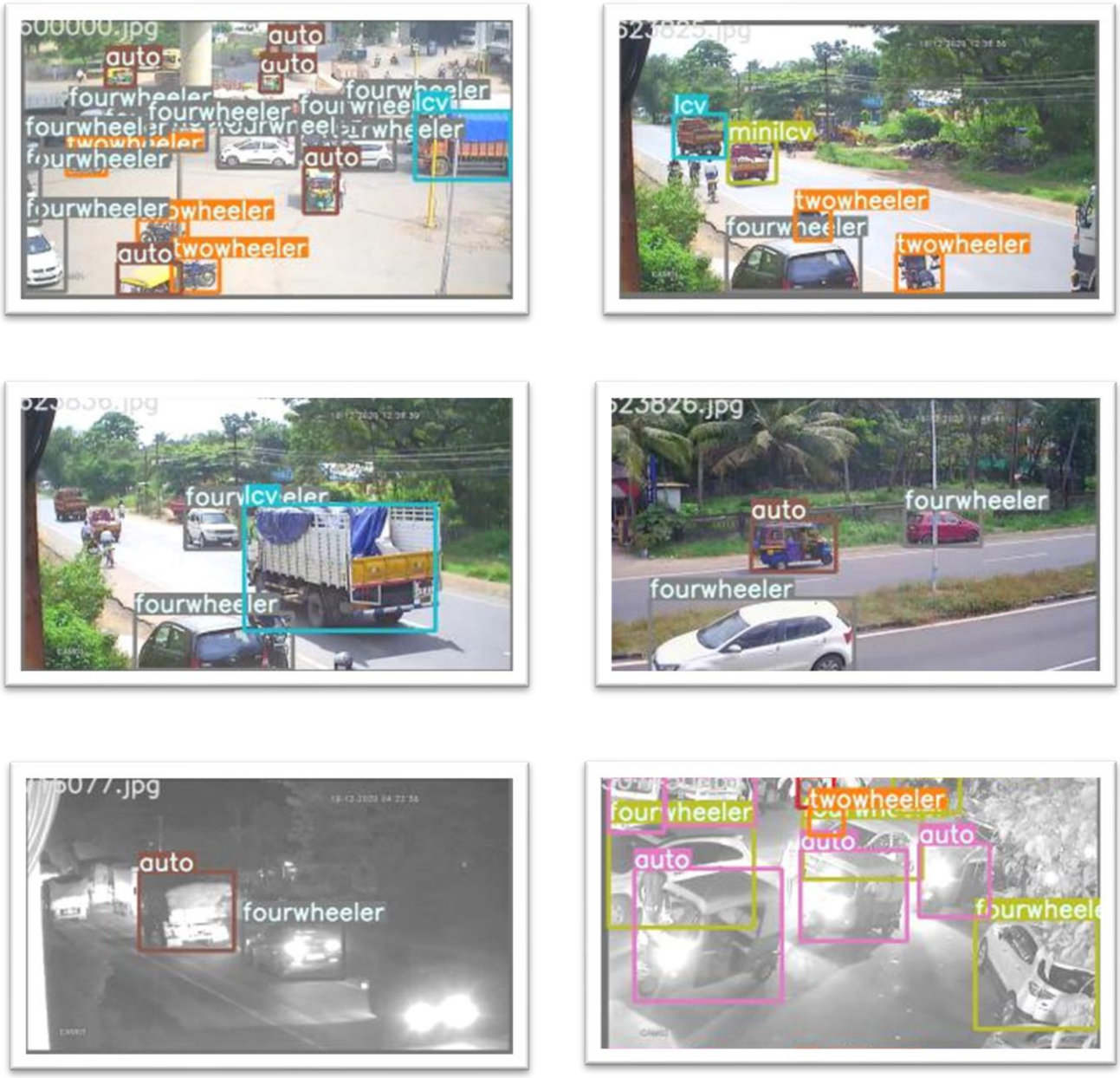
Total Annotated Day-Time Images: 33,165
Total Annotated Night-Time Images: 27,575

The data is pre-processed in Darknet '.txt' Yolo Format within the following 17 Classes using LabelImg Tool. Below are the Classes name mentioned along with their net count.

S.No	Vehicle Class Name	Day Time Count	Night Time Count
1.*	Cycle	4855	1.997
2.*	Two-Wheeler	74,016	26,799
3.	Cycle-Rickshaw	283	209
4.*	E-Rickshaw	710	778
5.	Handcart	350	62
6.*	Passenger Auto	44,492	14,614
7.*	Commercial Auto	4,024	496
8.*	Four-Wheeler	75,213	48,758
9.*	Mini Light Commercial Vehicle	8,463	2,805
10.*	Light Commercial Vehicle	8,010	3,120
11.	Mini Bus	895	313
12.*	Bus	3,311	1,915
13.	Two Axle Trucks	3,252	1,713
14.	Three Axle Trucks	1,655	888
15.	Four to Six Axle Trucks	2,112	1,034
16.	Heavy Class Machinery	2,326	9
17.	Tractor with Trailer	209	0

Table I : Represents the total Number of Annotations obtained

'*' represents classes included in a special set of 9 Classes.



Img II : Depicting the Labels of Dataset

DIGITAL IMAGE PROCESSING

I. Technique I : Contrast Enhancement

We have used OpenCV Library in Python to :

- Apply Histogram equalization for contrast improvement.
- Apply Contrast Limited Adaptive Histogram Equalization (CLAHE) to take care of over-amplification of contrast (to equalize image).

Image II: Application of Digital Image Processing technique I of night time image 1, night time image 2 and day time image 1 respectively.

Left Hand Side display the original Image, Right Hand Side display the enhanced Image.

The technique did enhance the contrast in both night and day time images. However, this led to enhancement of features in areas of low intensity and decrease of features in area of high intensity. Due to Extreme focus on contrast enhancement, additional noise in the form of white glare were imposed on night-time images.

II. Technique II : Glare Reduction

We have used OpenCV Library in Python to :

- Apply Gaussian Filtering (Gaussian Blur) to remove Gaussian Noise from the Image and Smoothen it.
- Threshold the Image above 200 Intensity value.
- Distinguish object boundaries using 'measure.label' from 'Scikit Learn' framework.
- Masking white patches, enhacing the contrast and the brightness surrounding the patches of high-intensity pixels.
- Finally, denoising the image again while adjusting contrast using 'fastN1MeansDenoising'.

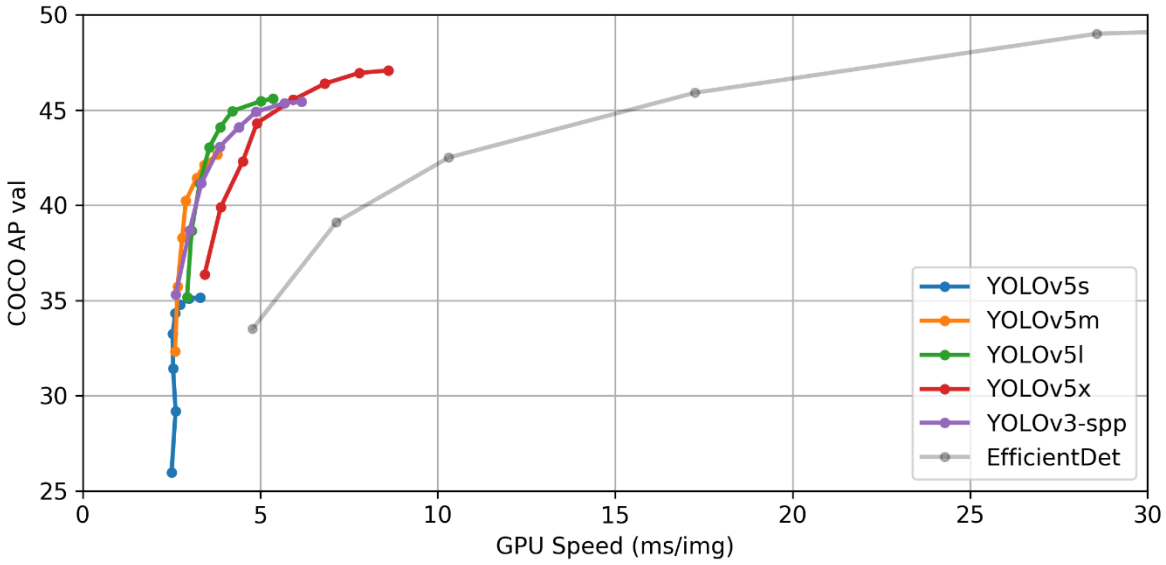
Image III: Application of Digital Image Processing technique II of night time image 3 and day time Image 2 respectively.

Left Hand Side displays the original Image, Right Hand Side displays the enhanced Image.

The technique enhanced the image contrast as well as enhanced the Glare in nigh time Images. However, the enhancements were very low to consider and are image dependent.

OBJECT DETECTION AND CLASSIFICATION

Yolov5 (You Only Look Once) is an open-source object detection model. We have chosen Yolov5 because it outperforms all previous versions of Yolo and got near to EfficientDet AP with higher FPS (Processing Speed).



Img IV: Comparaison of Yolov5 Models with Yolov3 and Efficient Det AP.

It is designed to create features from input images and then feed these features through its architecture system to draw boxes around objects and predict their classes. It has an end-to-end differentiable network.

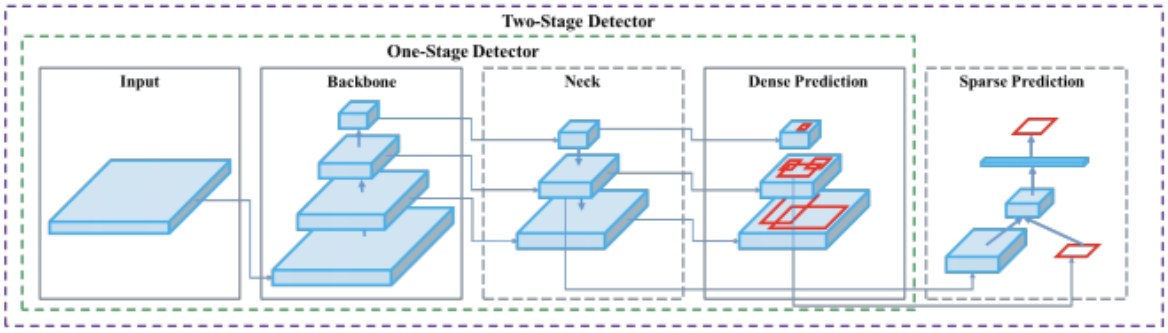
The Yolo network consists of three main pieces:

1. Backbone: A convolutional neural network that aggregates and forms image features at different granularities.

2. Neck: A series of layers to mix and combine image features to pass them forward to prediction.

3. Head: Consume features from neck and takes box and class prediction steps.

The architecture is implemented using PyTorch Framework.

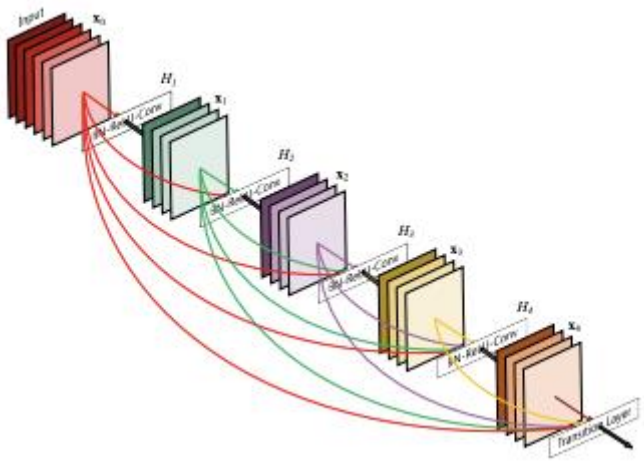


Img V : Object detection process followed in Yolov4.

Yolov5 automatic augments the data into various forms using its Data-Loader. It makes three kinds of augmentations: scaling, colour space adjustments, and mosaic augmentation (Helps to solve "small object problem").

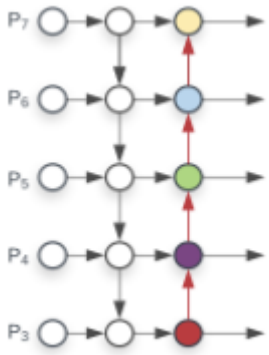
In order to make box predictions the Yolo network predicts bounding boxes as deviations from a list of anchor box dimensions. The anchor box is learned based on the distribution of bounding boxes in the custom dataset with K-means and genetic learning algorithms.

Yolov5 implements the CSP Bottleneck to formulate image features – Cross stage partial networks for convolutional neural network backbone. The CSP models are based on DenseNet and was designed to connect layers in convolutional neural networks to alleviate the vanishing gradient problem, to provide support to feature propagation, encourage network to reuse features and reduce number of network parameters.



Img VI : CSP network backbone visualization

For feature aggregation Yolov5 implements the PA-NET neck. The neck takes the feature layer from the CSP backbone as input.



Img VII: PA-NET neck representation

Also, compared to other object detection frameworks, Yolov5 provides following quality updates:

1. Easy to use: The repository comes handy with self-integrated image augmentation techniques.

2. Easy Install: Only requires the installation of torch and some lightweight python libraries.

3. Fast Training: Yolov5 models train extremely quickly which helps cut down experimentation costs as we build our model.

4. Inference Ports that work: We can infer with Yolov5 on individual images, batch images, video feeds, or webcam ports.

5. Intuitive Layout: File folder layout is intuitive and easy to navigate while developing.

6. Easy translation to Mobile: We can easily translate Yolov5 from PyTorch weights to ONXX weights to CoreML to IOS.

However, there are still debates among the better architecture amongst Yolov4 and Yolov5. New results show that the performance of Yolov5l is similar to that of Yolov4 (CSP, Leaky) and hence, we might provide comparative results on both models later in the study.

'*Table II*' represents the details regarding the training experiments carries out. All experiments were carried out using **Batch Size** of **16 Images**, Image **Dimensions** of **640x640**, **100 epochs** and train-test ratio equal to **0.3**.

Experiments with '*' show that the model was trained on 9 Classes. (Referring from '*Table I*' as Class Index Numbers: 1, 2, 4, 6, 7, 8, 9, 10, 12).

Experiments with '**' show that the model was trained on Day 17 Classes and Night 9 Classes Mix.

Exp No.	**Data-Set**	**Config.**	**Methodology**
1.*	Night	L	Trained from Scratch on Yolov5l Coco-dataset weights.
2.	Day	M	Trained from Scratch on Yolov5m Coco-dataset weights.
3.**	Day Night Mix	L	Trained from Scratch on Yolov5l Coco-dataset weights.

Table II : Representing various Training experiment details

YOLOV5 TRAINING CONCLUSIONS

The following table depicts the result of all Training Experiments along with their testing results on Day Time and Night Time Test Set.

Testing Details:

- Test Set Day: The whole **Day Time** dataset is taken as test-set containing **33,165** Images.
- Test Set Night: The whole **Night Time** dataset is taken as a test-set containing **27,575** Images.
- Testing Model confidence was set as **0.85**
- Testing Area of Intersection over Union (IOU) was set as **0.85**.
- The Standard to Measure the performance of the Experiment is **Weighted Standard Deviation * 100** from an Ideal hypothesized confusion matrix for which the accuracy is 100%, i.e. the matrix score for all diagonals is 1.0.

Exp. No.	Day Time Standard	Night Time Standard
1.*	96.106	60.438
2.	44.709	68.209
3.**	**38.171**	**67.392**

Table III : Depicting the Standards obtained on Day-Time and Night-Time data by all Experiments Applied.

Thus, from the testing shown on '*Table III*' results we can infer the above **Conclusions**:

1. Comparing results obtained from '*Experiment 1*' and '*Experiment 2*' we can infer that Day-Time conditions are very different from Night-Time conditions. Also, Results obtained from '*Experiment 1*' show that Model trained on Night-Data perform

extremely poor on Day-Time data.

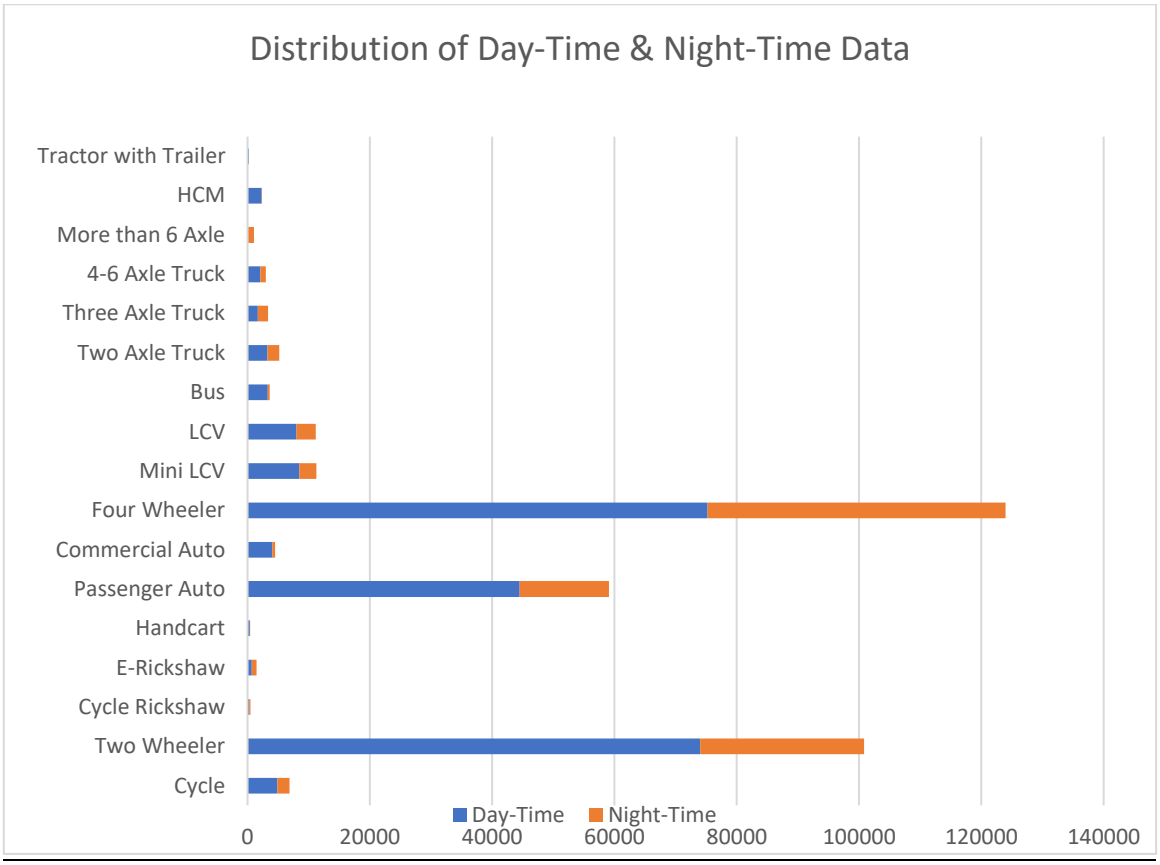
Also, we see that Day-Time Models can perform much better on Night-Time models, as compared to performance of Night-Time models on Day-Time. This, indicated the presence of large amount of Noise in night-time conditions.

2. A **Mix of Day and Night Time data provided improved standard** for both the conditions, suggesting that augmenting low-resolution noisy data with enhanced quality data, may help in improving the accuracy.

3. The method for dividing dataset into batches and training each batch separately due to low hardware specifications is not recommended. This would result in poor accuracy in case the data annotations are not well distributed among annotation files.

YOLOV5 TRAINING RESULTS

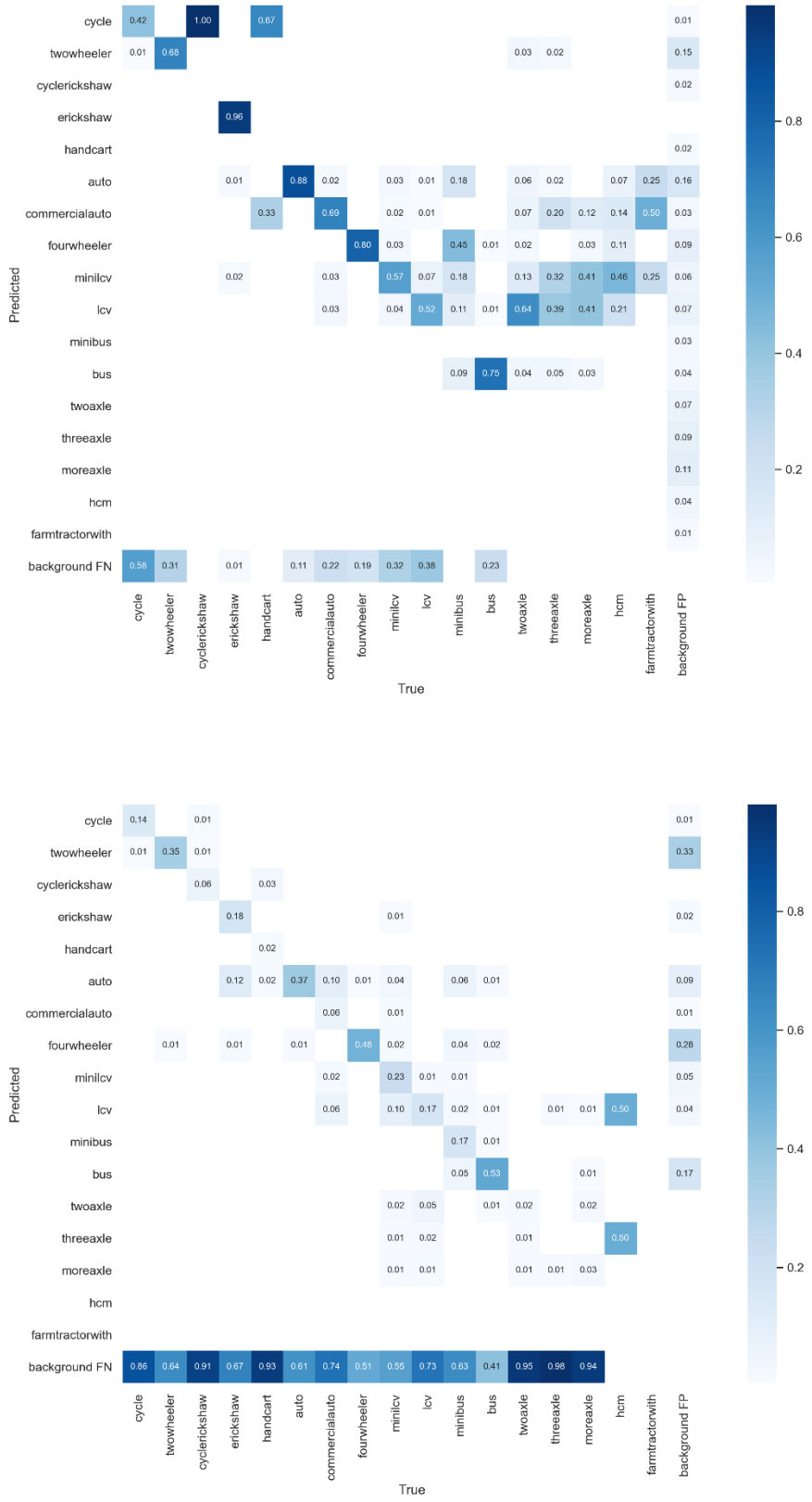
For Model L of Yolov5 the best results are obtained on '*Day Night Mix*' Dataset. This may be due to the fact that good resolution and less noisy data when present in appreciable quantity may improve the accuracy obtained on Low-Quality noisy images . (Refer to '***Experiment 3***')



Bar Chart I : Depicting the distribution and Count of 'Day Night Mix' Dataset.

Accuracy obtained by the trained Model on **IOU** of **0.7**, and Confidence of **0.7**:

1. Day Time Weighted Accuracy: **71.398%**
2. Night Time Weighted Accuracy: **40.313%**



Img VI : Top represents the Confusion matrix obtained on Day-Time test-set and Bottom represents the Confusion matrix obtained on Night-Time test-set.

OBJECT TRACKING

For Object Tracking purposes, we have used DeepSORT Tracking algorithm (Simple Online and Realtime Tracking with Deep Association Metric)

I. Simple Online and Realtime Tracking (SORT)

- Multiple Object tracking detection framework is used. Contrast to other batch-based tracking approaches it is targeted towards online tracking where only detections from previous and current frame are presented to tracker.
- It associates data, i.e it associates detections across frames in a video sequence.
- However, the issue regarding short- and long-term occlusions were ignored in SORT.

II. SORT with Deep Association Metric (DeepSORT)

- We integrate appearance information to improve the performance of SORT. Due to this extension we are able to track object through longer periods of occlusions, effectively reducing the number of identity switches.
- For motion estimation Kalman filter is used along with Hungarian method for data association in DeepSORT tracking.
- Kalman filter predicts estimated position of the track in next frame based on the motion of the track in previous frames.
- Hungarian method is used to solve the association between predicted Kalman states and newly arrived measurements.
- Cosine metric is used for measuring smallest cosine distance between i^{th} track and j^{th} detection in appearance space.
- By using simple nearest neighbour queries without additional metric learning, successful application of our method requires a well-discriminating feature embedding to be trained offline. This is trained by minimizing difference between same class and maximizing distance between different classes.

III. Technique for Tracking

- Due to dataset restrictions, we have used pre-trained weights trained on pedestrian traffic, acquired through Mars dataset.

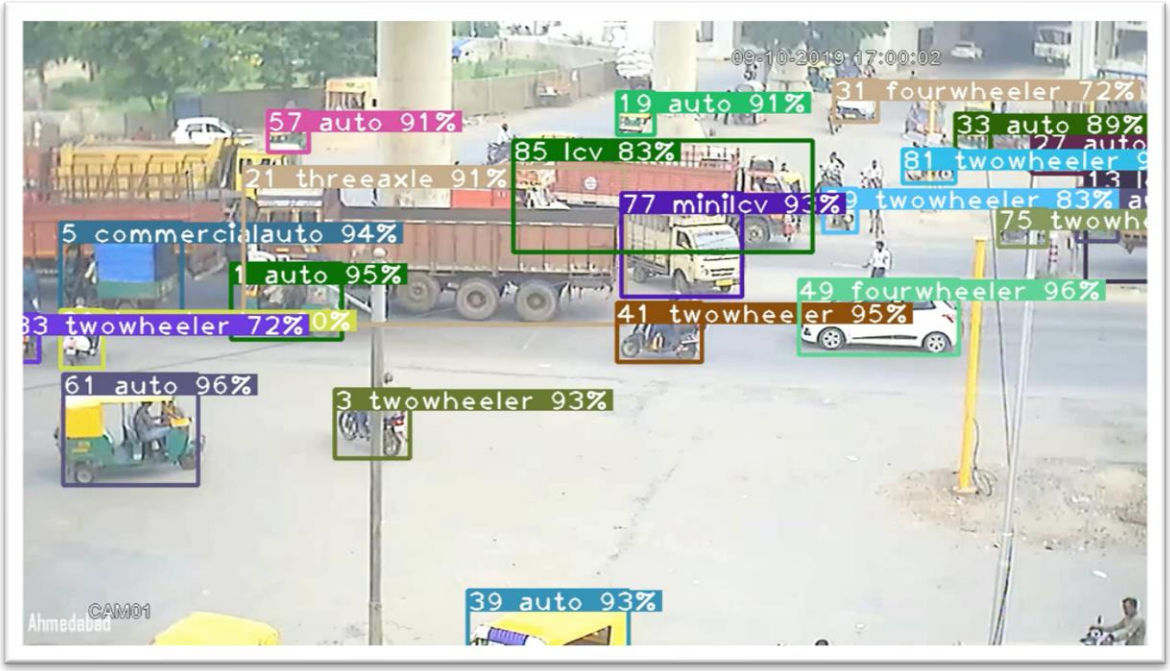
IV. Hyper-Parameter Tuning to our use case

DeepSORT is dependent on the above parameter which can be modulated as per the tacking use case.

- Max Distance (MAX_DIST): Measures smallest cosine distance between i^{th} and j^{th} detection in appearance space.
- Neural Network Budget (NN_BUDGET): It is a gallery of features for each track. It is a list with the characteristics of each time that track appeared.
- Max Age (MAX_AGE): The number of frames till which a id is kept in consideration.

Manipulating and testing out various hyper-parameter conditions for our use case, the best values for the parameters obtained are:

- MAX_DIST: 0.2
- NN_BUDGET: 100
- MAX_AGE: 70



Img : DeepSORT tracking vehicles in Real-Time on Day-Time and Night-Time Images.

CONCLUSION

Object detection and classification like Yolov5 and Yolov4 are able to provide appreciable accuracy on Day-Time vehicle tracking. However, accuracy obtained on Night-Time data by the models, are quite low due to conditions such as low resolutions, low illumination, high amounts of blur and noise. The above problem can however be tackled by increasing data and experimenting with various augmenting techniques.

DeepSORT tracking algorithm performs considerably well as compared to traditional OpenCV based tracking algorithms and is able to track objects in between large periods of occlusions.

FURTHER WORK – R&D

Creation and Cleaning for dataset will be performed to increase accurately marked data for classes, of which accuracy is below **70%**.

REFERENCES

[1]. *LabelImg 1.4.0*, Python Package Index.org

[2]. *Opencv-python 4.5.1.48*, Python Package Index.org

[3]. *Yolov5 by Ultralytics LLC*, Pytorch.org

[4]. *Simple Online and Realtime Tracking with a Deep Association Metric*, arXiv

[7]. *Multi-class_Yolov5_DeepSort_pytorch : WuPedin*, GitHub

[8]. *Yolov5 New Version – Improvements And Evaluation*, Roboflow Blogs