BDD100k Dataset Analysis

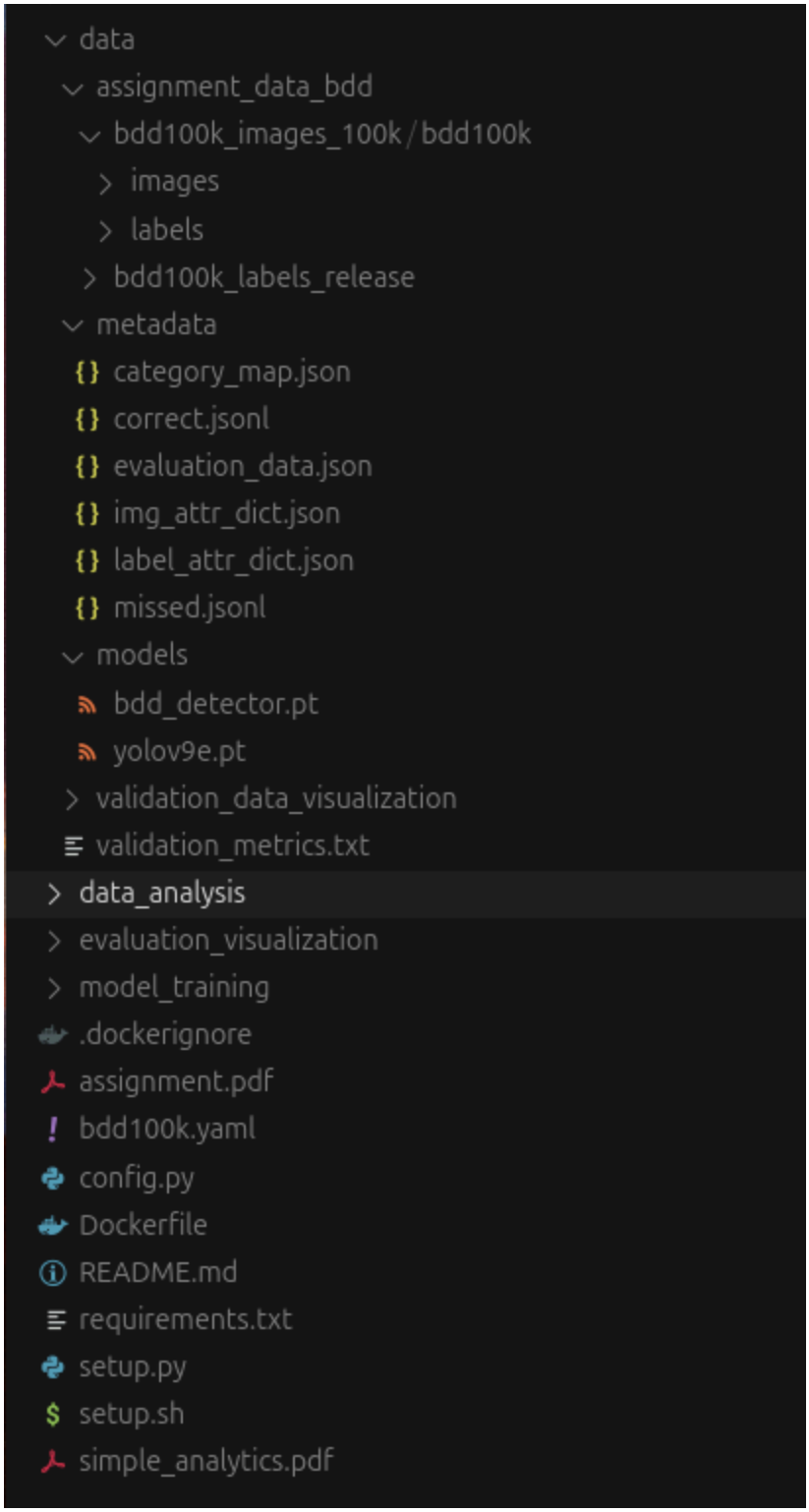
Name: Surya Prasath Ramalingam
Email: suryaprasathram@gmail.com

Data Analysis

Dataset Loading and Processing

The BDD 100k dataset follows the below structure:

```
bdd100k/
├── /../images/../                  # All images (100k total)
│   └── 100k/                        # Full set (100k images for training/validation)
│       ├── train/
│       ├── val/
│       └── test/
├── /../labels/../                  # Annotations in JSON format
       ├── train_annotations.json
       ├── val_annotations.json
       └── det_test.json
```



The annotation files for the training and testing data are huge (1.5GB and 200MB, respectively). This can cause slow processing and retrieval of data. To make this more efficient, we can convert them into the JSONL format, where each annotation entry is written as a single line in a

text document. Since the structure is explicitly built from each line during runtime, this reduces the time required to load large datasets.

Metadata and other generated data

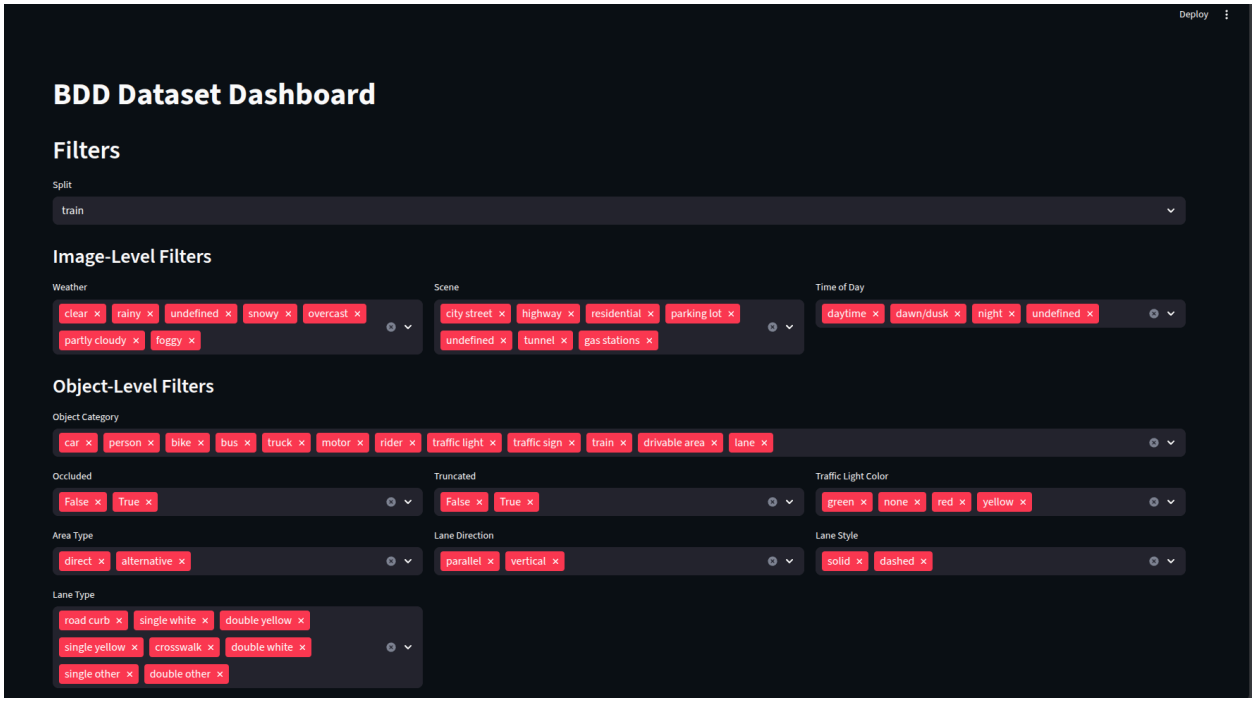
- Class mapping (class → index)
- Validation set evaluation metrics for analysis
- Validation set predictions for analysis
- YAML configuration file for training detection models
- Alternate label formats for training detection models
- A default configuration file (`config.py`), primarily holding dataset paths for all tasks

To refer to the data structure and parser: `data_analysis/reader.py`

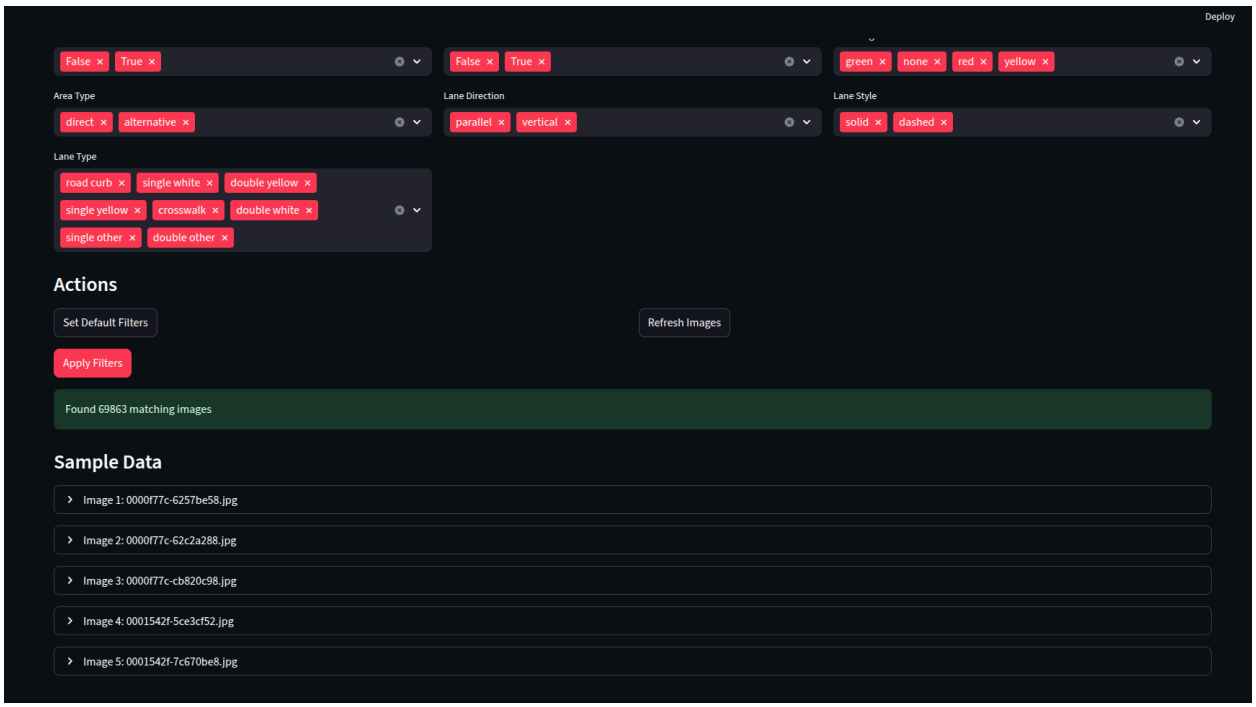
For performing data analysis, both quantitative and qualitative, a dashboard is created and shown in the next section.

Dashboard

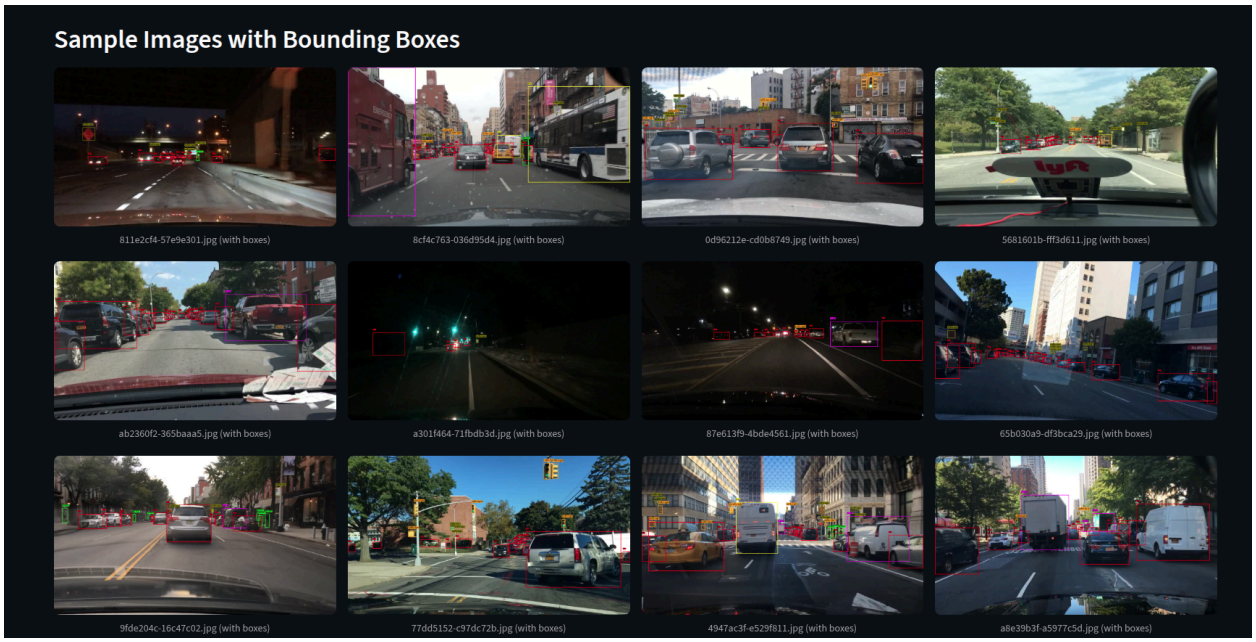
The dataset dashboard is an interface that allows both quantitative and qualitative analysis with high flexibility. The filters are divided into two sections based on attributes: image-level and object-level. Users can apply their own filters based on their preferences.



After applying the filters, matches are found. There are three buttons: one for applying filters, one for clearing all filters and resetting them to default values (to allow everything), and one for refreshing images, which will be shown in the next section.

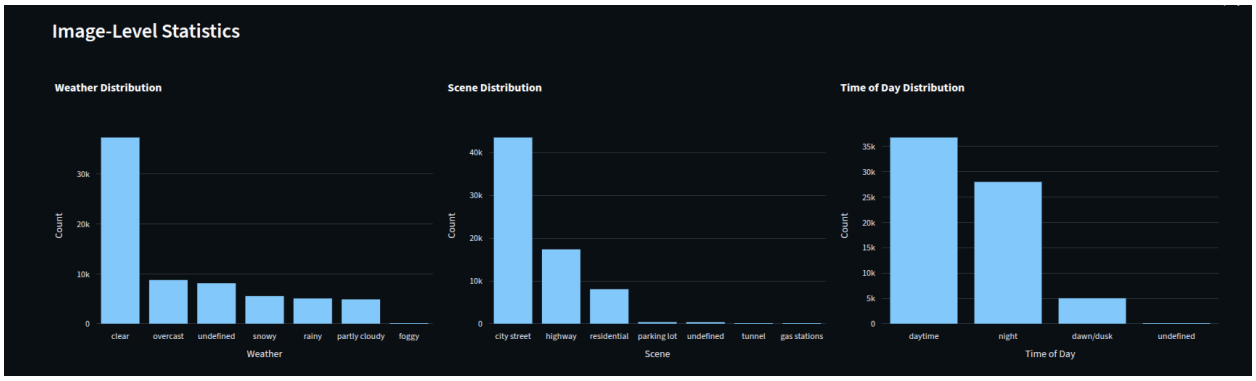


The images shown here are randomly selected, with bounding boxes drawn on them, and a button is provided to open the image in full screen for closer observation.

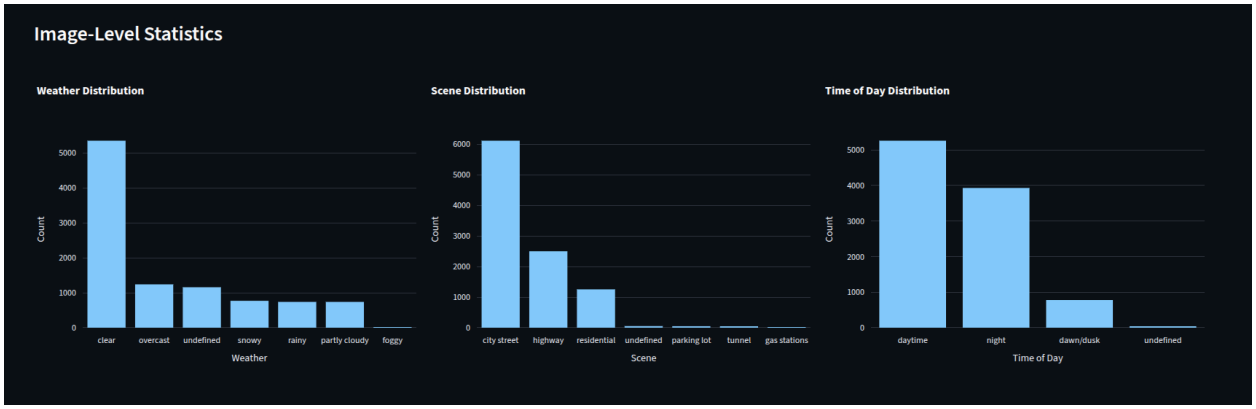


Then, image-level statistics and label-level statistics are generated. These are shown side by side along with the validation split (when applied in the filter section).

Train:



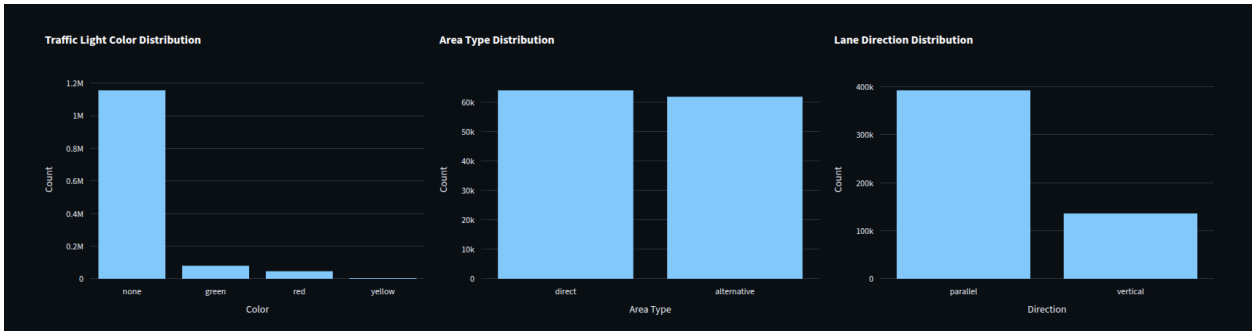
Val:



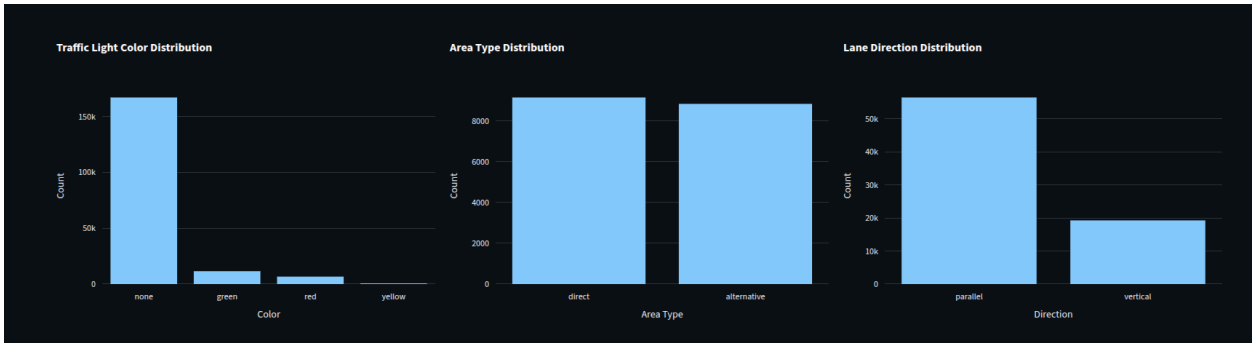
Train

Val:

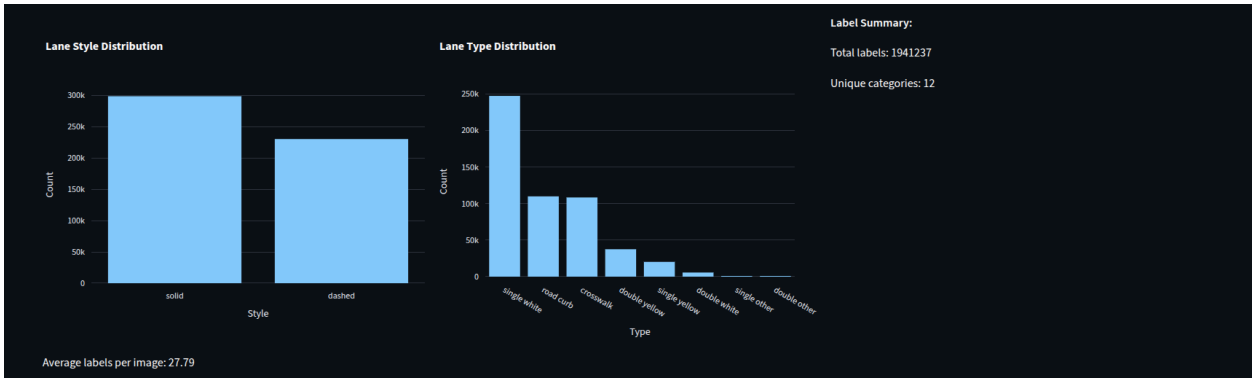
Train:



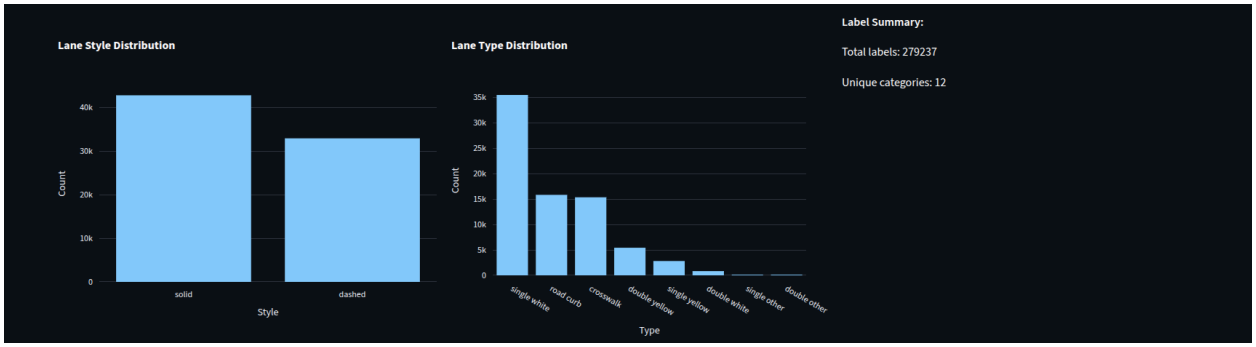
Val:



Train:

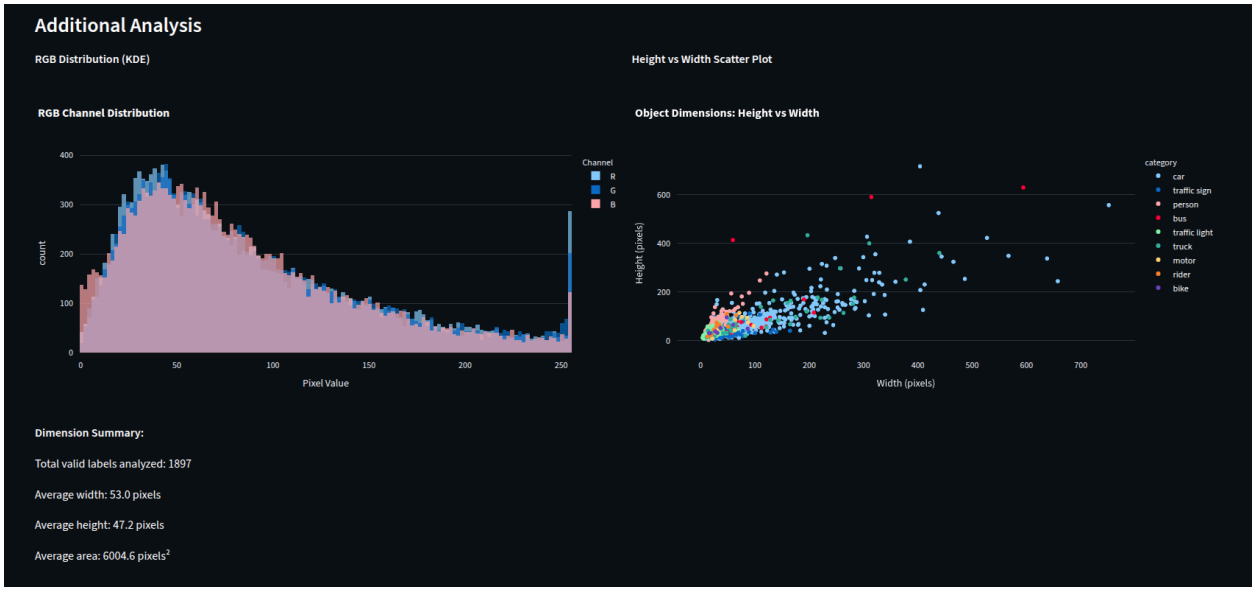


Val:

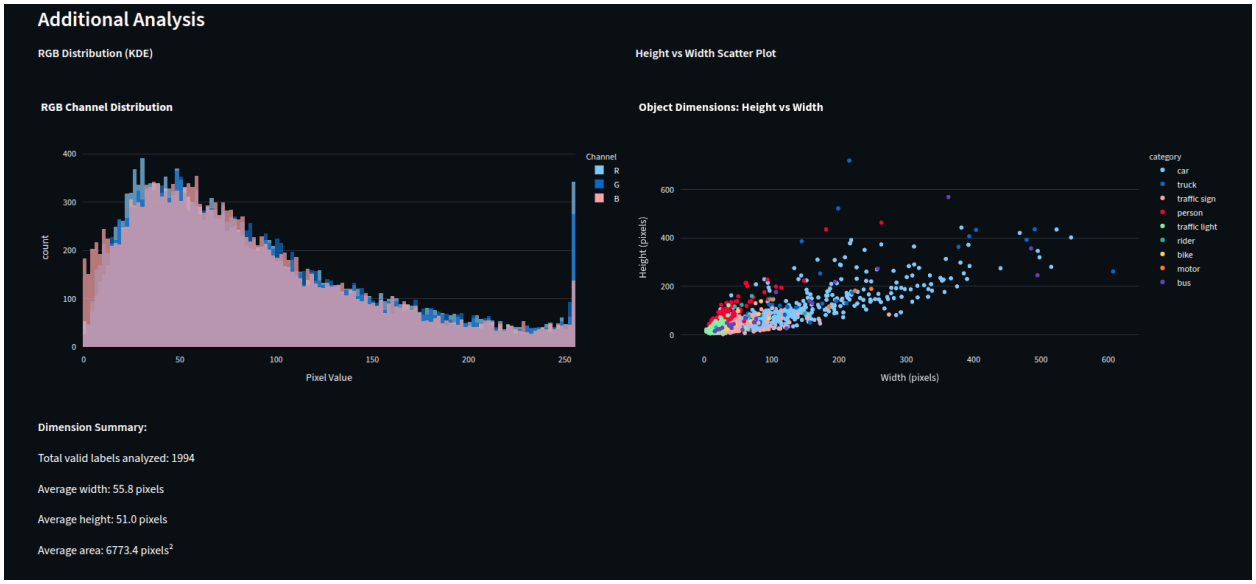


Additionally, plots such as RGB channel distribution, height vs. width comparison, and average area are demonstrated for all classes.

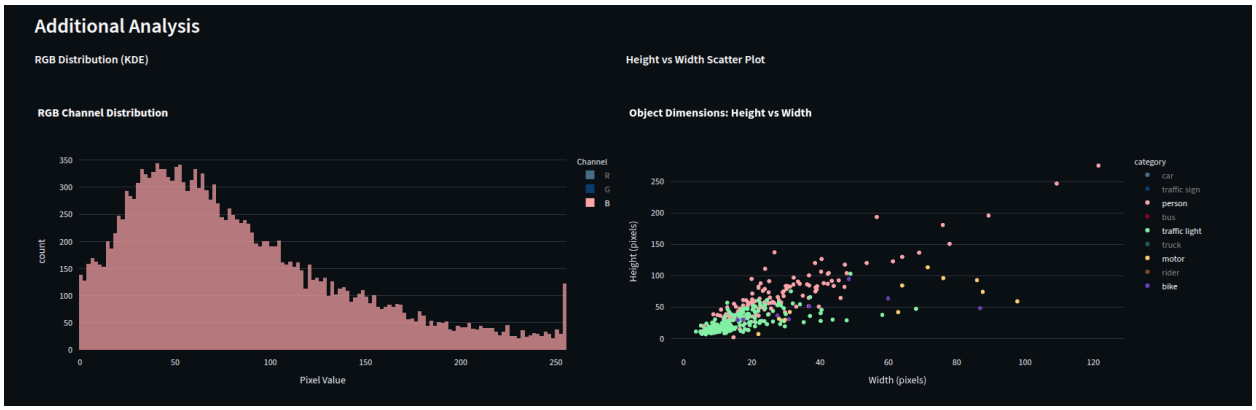
Train:



Val:



The plots are also interactive, giving users the flexibility to interpret them effectively.



The changes are in real time—that is, modifying a filter will automatically adjust the rest of the statistics. For example, let’s say we want to focus on occluded bikes at night.

Filters

Split

train

Image-Level Filters

Weather

rainy × foggy × overcast × clear × snowy × partly cloudy × undefined ×

Scene

city street × highway × residential × parking lot × undefined × tunnel × gas stations ×

Time of Day

night ×

Object-Level Filters

Object Category

bike ×

Occluded

True ×

Truncated

False × True ×

Traffic Light Color

green × red × yellow × none ×

Area Type

direct × alternative ×

Lane Direction

parallel × vertical ×

Lane Style

solid × dashed ×

Lane Type

road curb × single white × double yellow × single yellow × crosswalk × double white × single other × double other ×

Actions

Set Default Filters

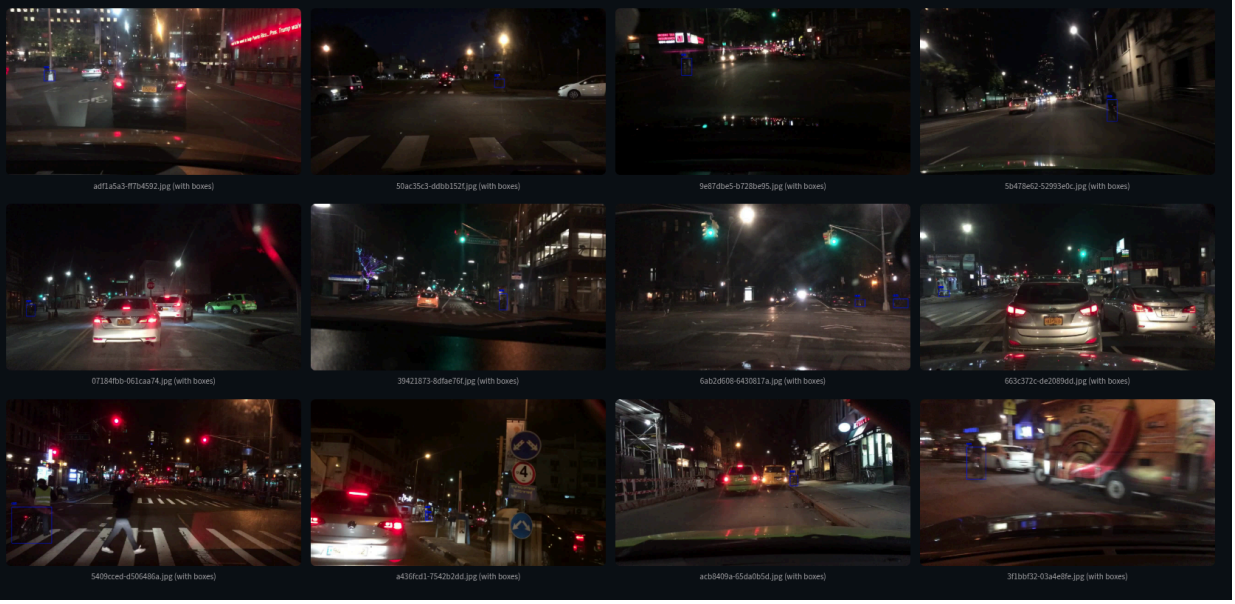
Refresh Images

Apply Filters

Found 223 matching images

Sample Data

Sample Images with Bounding Boxes

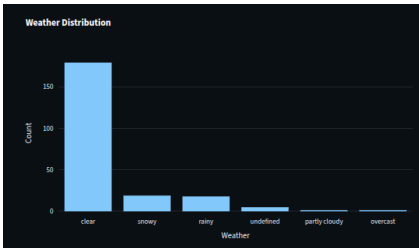


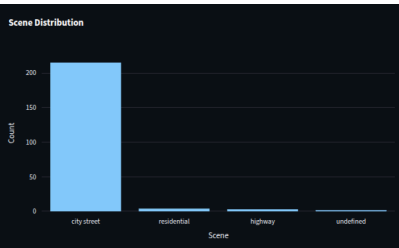
adf1a5a3-ff7b4592.jpg (with boxes) 50ac35c3-ddbb152f.jpg (with boxes) 9e87dbe5-b728be95.jpg (with boxes) 5b478e62-52993e0c.jpg (with boxes)

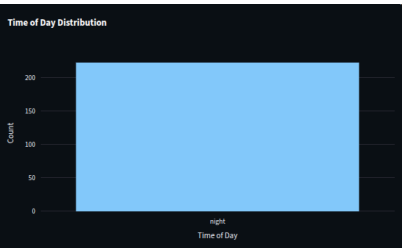
07184fbb-061caa74.jpg (with boxes) 39421873-8dfae76f.jpg (with boxes) 6ab2d608-6430817a.jpg (with boxes) 663c372c-de2089dd.jpg (with boxes)

5409cced-d506486a.jpg (with boxes) a436fcd1-7542b2dd.jpg (with boxes) acb8409a-65da0b5d.jpg (with boxes) 3f1bbf32-03a4e8fe.jpg (with boxes)

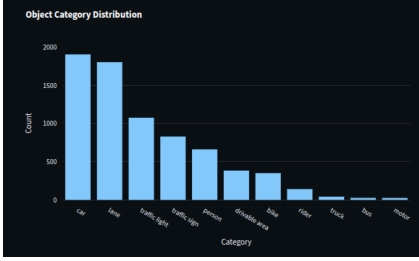
6ab2d608-6430817a.jpg (with boxes)

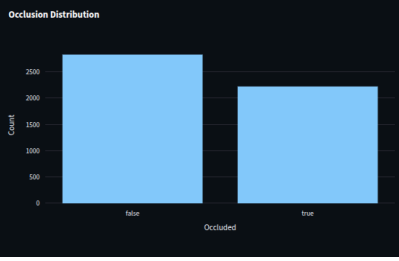


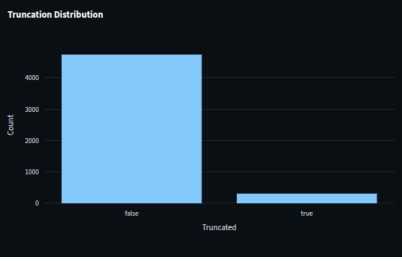


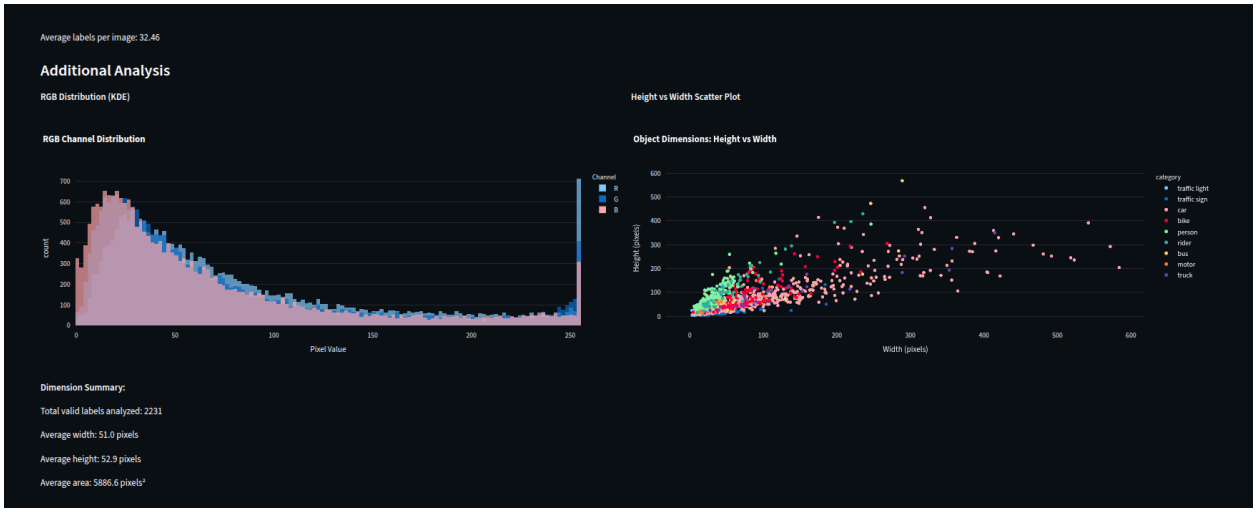


Label-Level Statistics









Since this is a very flexible dashboard, examples can be visualized at both the image and label levels.

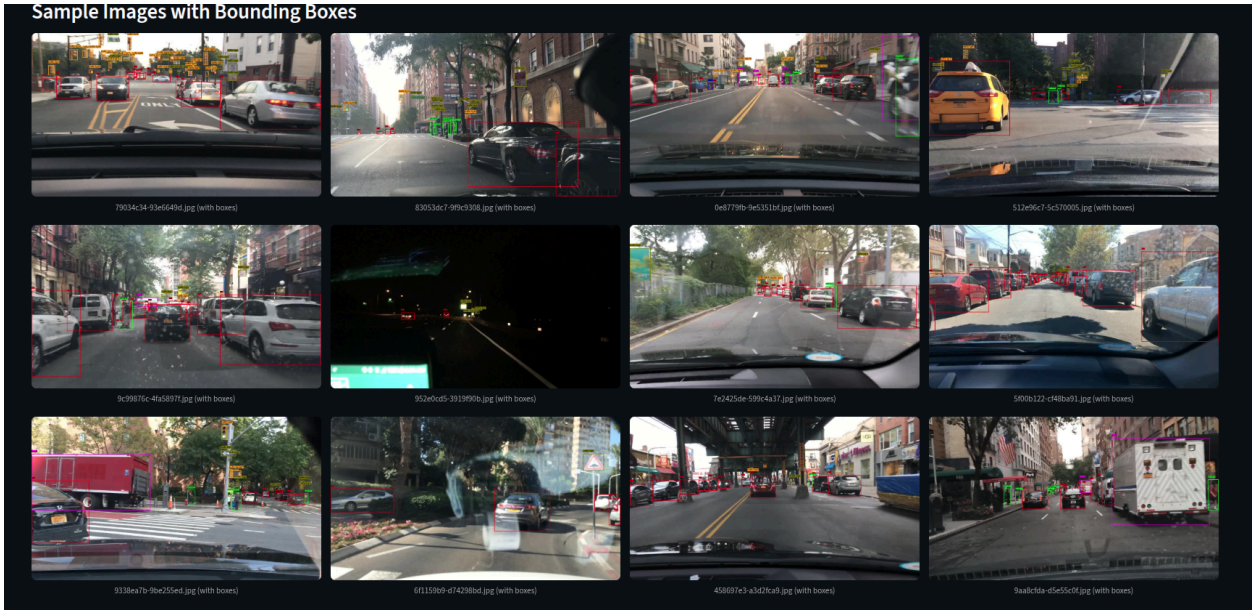
Interesting analytics

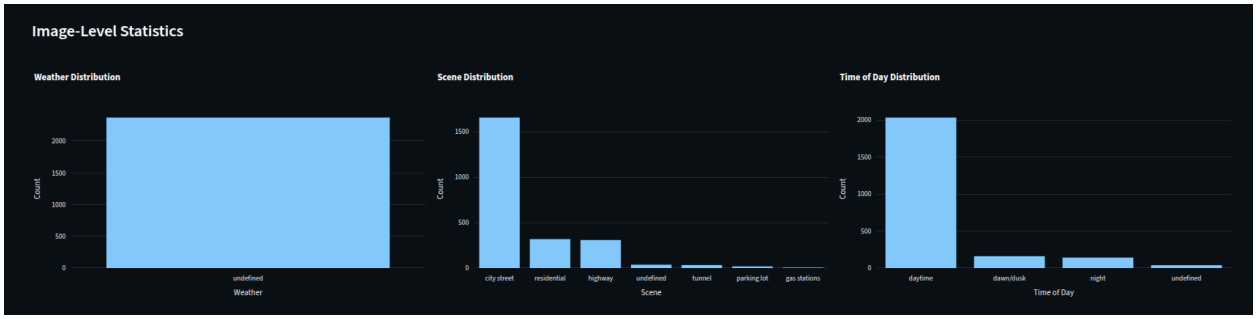
Here are some of the interesting analyses performed using this tool:

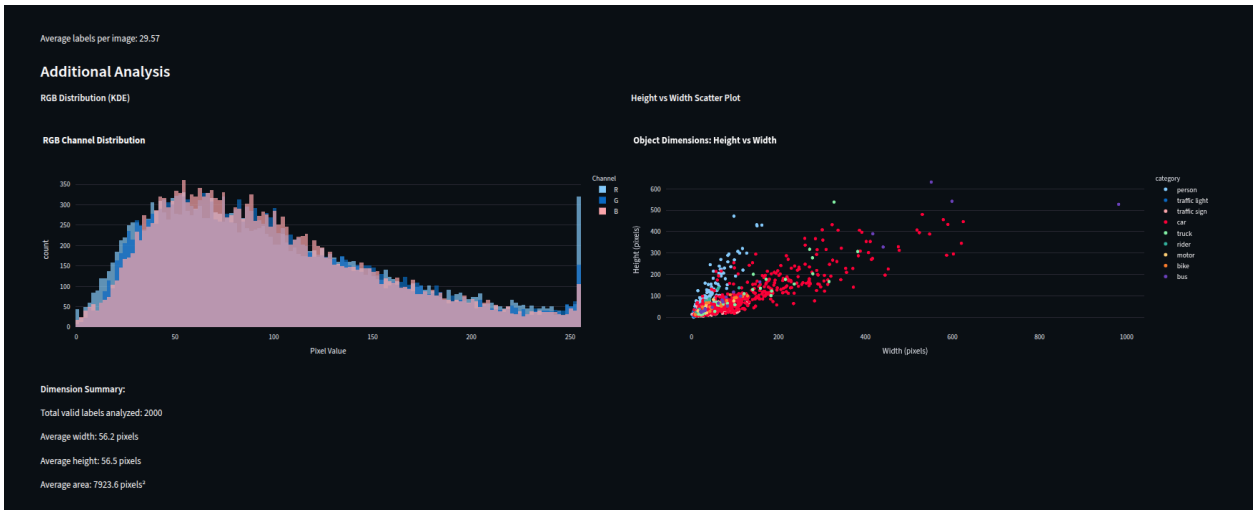
- The training and validation datasets are very similar in all attributes, both image-level and label-level. This clearly indicates that they fall under the same distribution, are randomly sampled, and, due to their large size (100k images in total), this similarity is evident.

- The undefined values for many attributes occur due to several reasons:

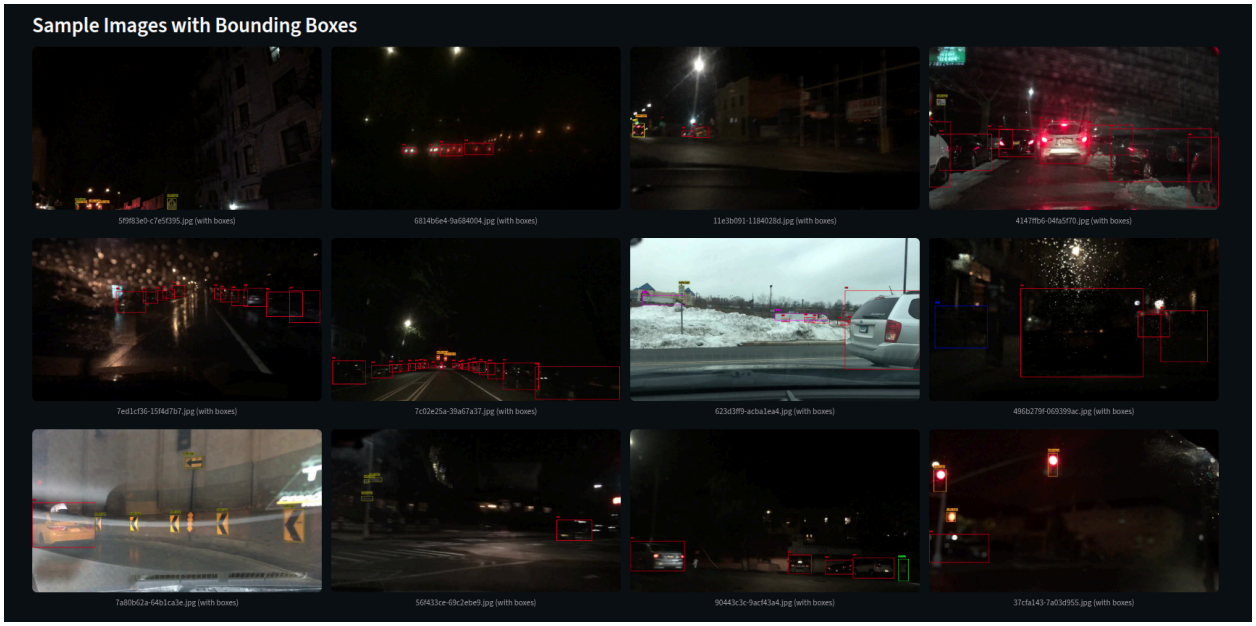
    - The sky is blocked by infrastructure, making it impossible to infer.
    - The vehicle is inside a tunnel.
    - It is too dark to determine.

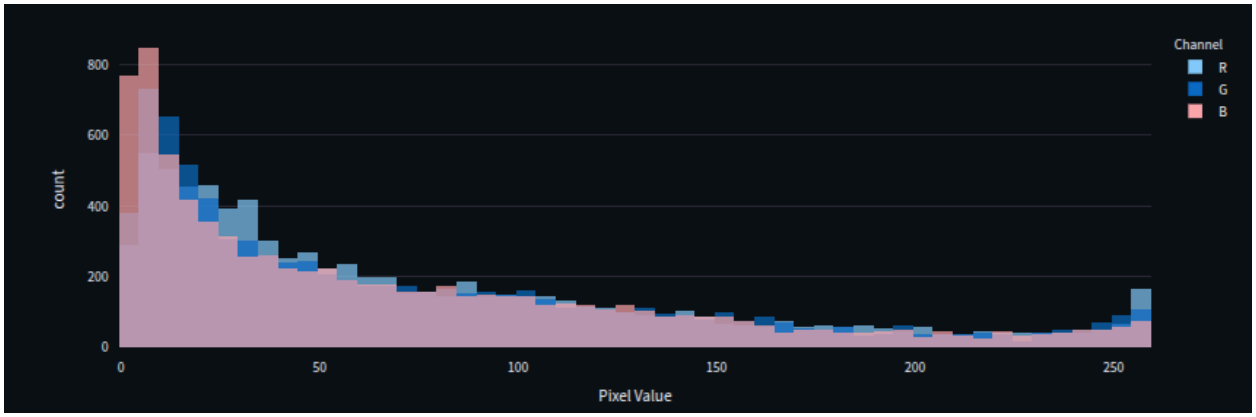




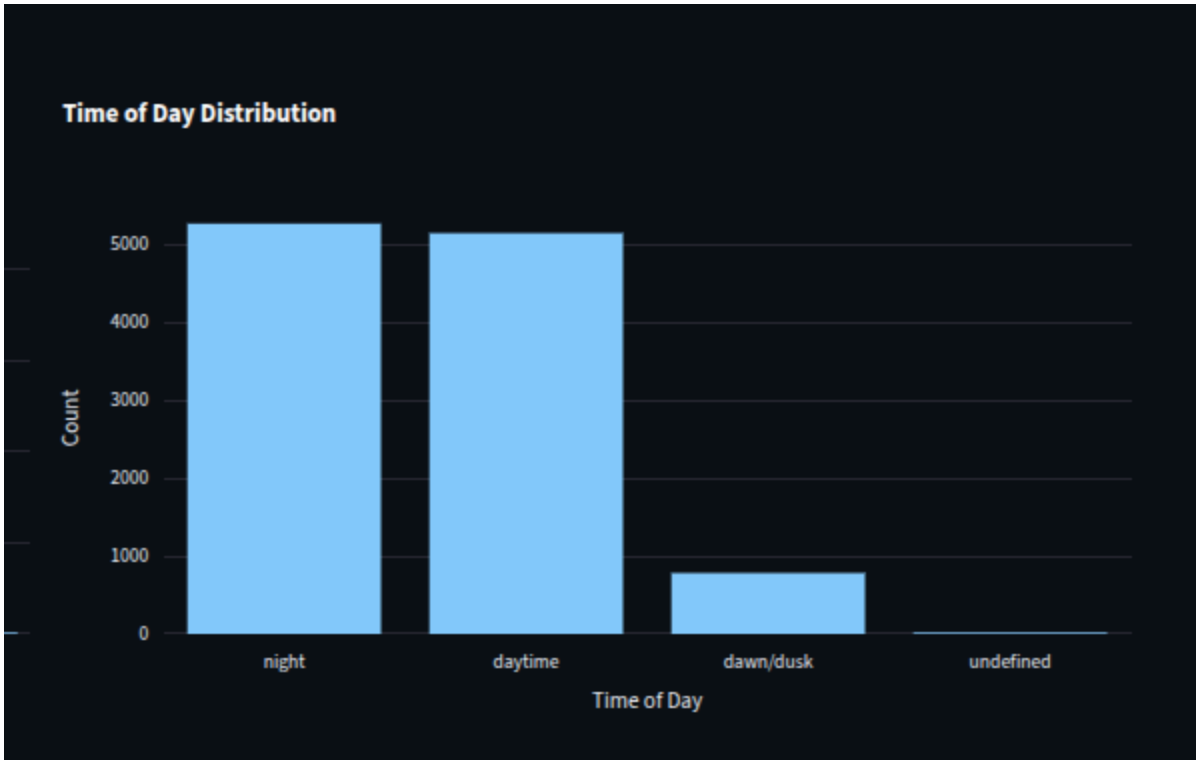


- Undefined location is mostly due to nighttime, when nothing is visible. The RGB values skewing toward zero confirm this.



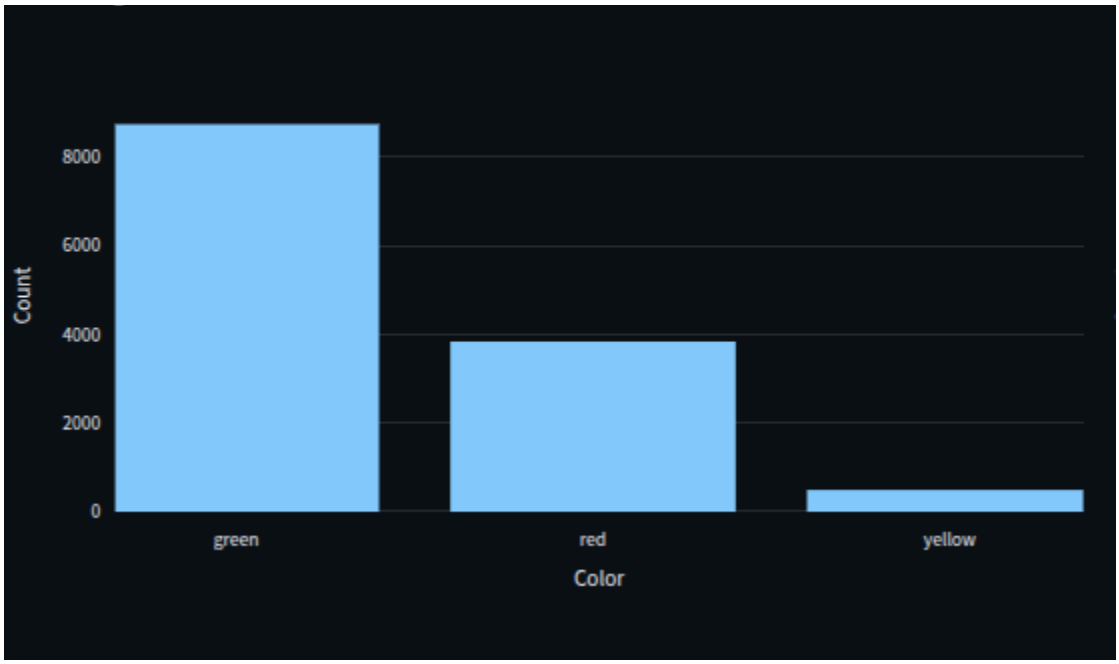


- The labeling quality for traffic lights is good, and surprisingly, there are an equal number of traffic light images in both daytime and nighttime.

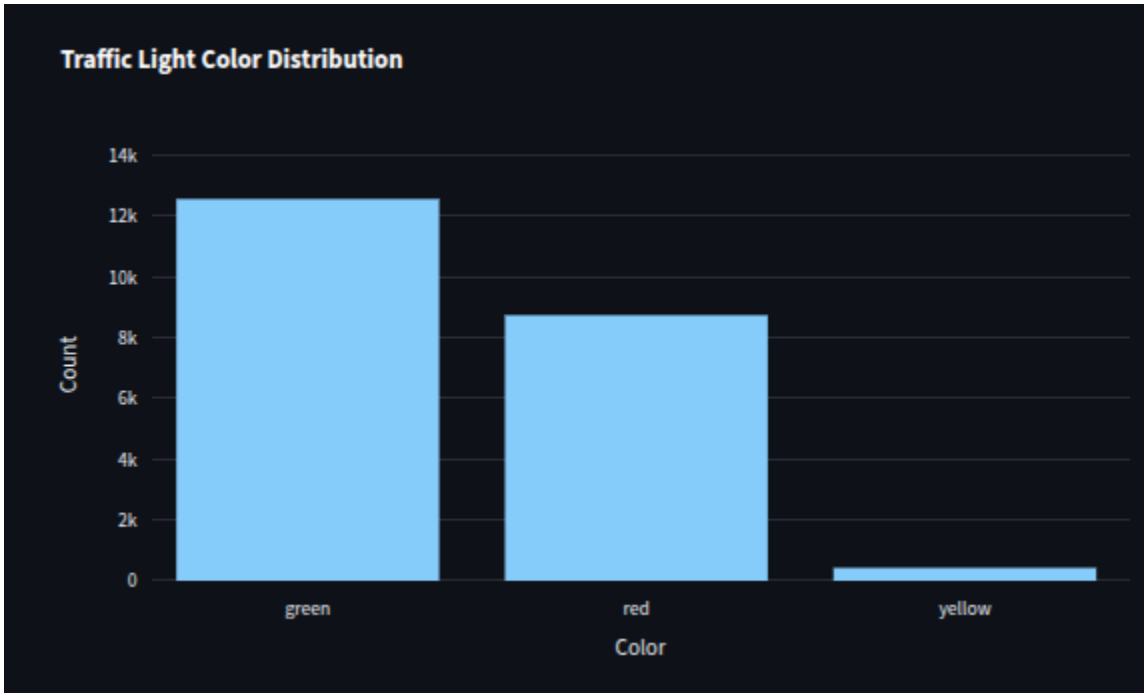


- There are, on average, 29.74 objects detected during the daytime, while at night there are 25.22 objects detected. Since the values are close, this implies that the dataset is carefully annotated.
- At night, red lights are much more visible compared to the daytime.

Day:



Night:



Apart from this, some general trends from the dataset are:

- The dataset is highly skewed, with several image-level attributes being one-sided. This can result in a loss of generalizability in models trained on this dataset.
- There is a significant class imbalance, with cars being the most detected and trains being the least.
- There is almost an equal share of occluded and non-occluded objects, making this dataset well-suited for training models robust to occlusion.
- There is also a good level of partitioning for lane types, which is beneficial for training lane detection models.
- The average area of all objects is around 6004 pixels, which is approximately 0.6%—a fairly small percentage.
- The images vary in resolution, blur levels, contrast, brightness, locations, zoom levels, and relative car positions, making this a highly diverse dataset.

Model

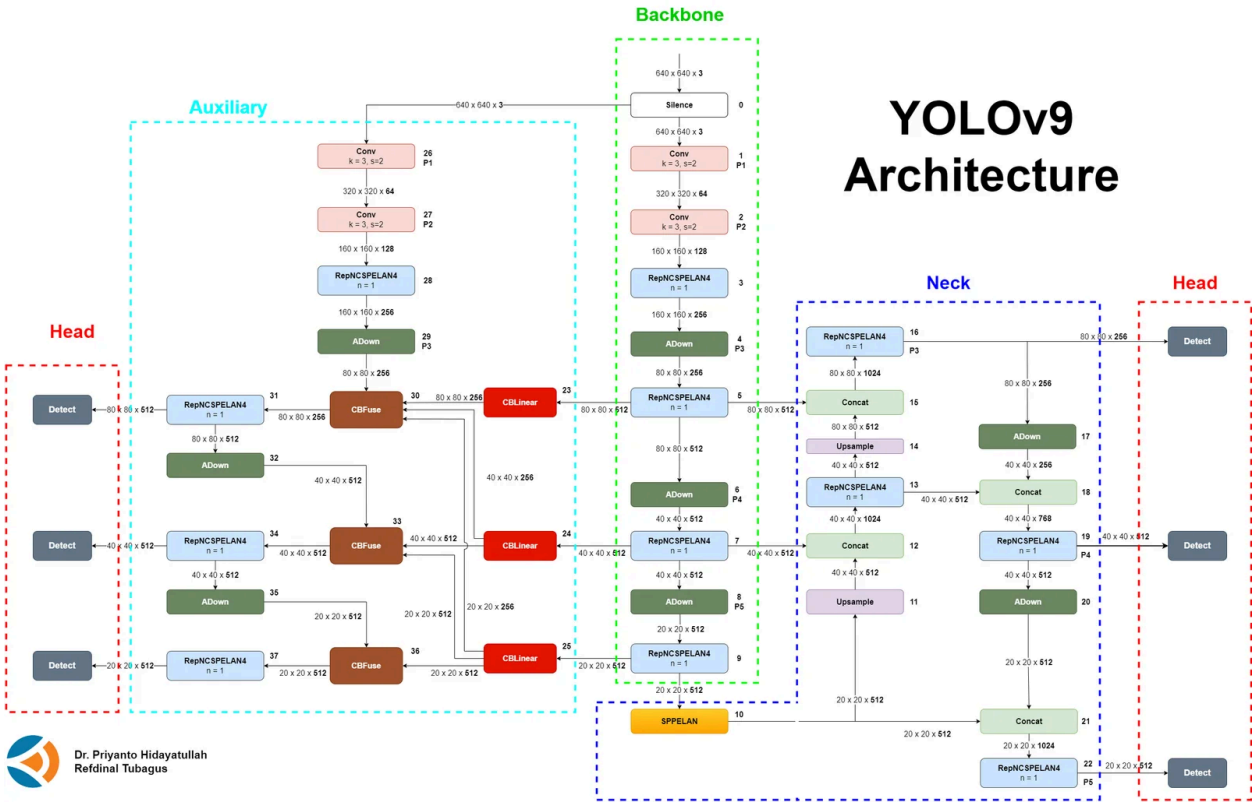
Selection of Model

For this task, the **YOLOv9e** model is selected as the best choice. The reasons are:

- The model is state-of-the-art in the MS COCO object detection task.
- It is relatively small in size compared to foundational models like DINOv3.
- Since we are not using the model for strict real-time operations, we can trade off higher latency for improved accuracy.
- It has two significant components that boost its performance
    - PGI : Programmable Gradient Information
    - GELAN : Generalized Efficient Layer Aggregation Network
- These two components ensure that information that is generally propagated through a deep learning network is not lost, as they preserve essential data for better gradient update.

The model is taken from the Ultralytics repository, which provides a tightly integrated wrapper around PyTorch functions. As a result, training the model directly in PyTorch becomes difficult, since the framework enforces the use of its wrapper functions. However, I have written the dataloader code in the `model_training/dataloader.py` file, which efficiently retrieves images and labels from the BDD100K dataset.

The script `model_training/yolo_train.py` can be used to train the **YOLOv9e** model on the **BDD100K** dataset. Training is configured through a YAML file along with alternate text labels, allowing the Ultralytics library to use the dataset directly. Due to computational constraints, the training is limited to **1 epoch**, with the first **28 layers of the backbone frozen** during training.



Validation and Evaluation

- **mAP:** 0.2179
- **mAP@50:** 0.4053
- **mAP@75:** 0.2020
- **mAP (small):** 0.1065
- **mAP (medium):** 0.3435
- **mAP (large):** 0.4463
- **mAR@1:** 0.1508
- **mAR@10:** 0.2877
- **mAR@100:** 0.3113
- **mAR (small):** 0.1906
- **mAR (medium):** 0.4630
- **mAR (large):** 0.5257

After training for just one epoch, the model achieves an overall mAP of 0.22, with mAP@50 of 0.41. This suggests that the model is already capable of generating predictions that overlap well with the ground truth at lower IoU thresholds. Performance, however, varies significantly across object sizes. Large objects are detected much more reliably (mAP_large 0.45, mAR_large 0.52)

compared to small objects (mAP_small 0.10, mAR_small 0.19). Medium objects fall in between, with relatively better results (mAP_medium 0.34, mAR_medium 0.46).

The drop from mAP@50 to mAP@75 indicates that localization is still poor. Recall is also low: around 15% with only one prediction per image, and still under 32% with multiple predictions. This shows that the model is missing many objects, especially smaller ones. Nevertheless, the fact that it is already detecting large objects somewhat reliably after just one epoch suggests that the model is beginning to learn effectively, though it has not yet converged.

Now, visualizing examples using the same dashboard (check README.md for instructions on how to run):

Showing some correct predictions:

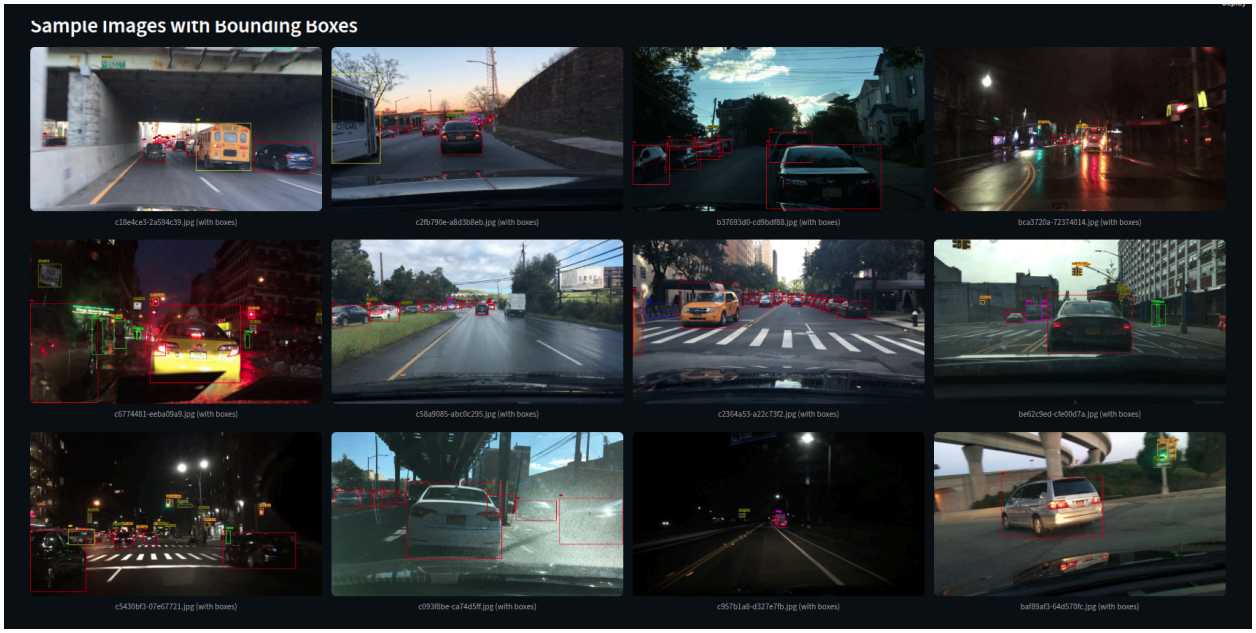
Next, showing some statistics on the correct predictions:

The plots show that the model is able to generalize to predicting the classes correctly. This is clear, since the model has converged the classification loss significantly. Additionally, it has no problem predicting at day or night, in different scenes etc.

Missed: 5000:3400
Missed: 5290:3900 (difference is huge) : pattern, at dark times, difficult to detect

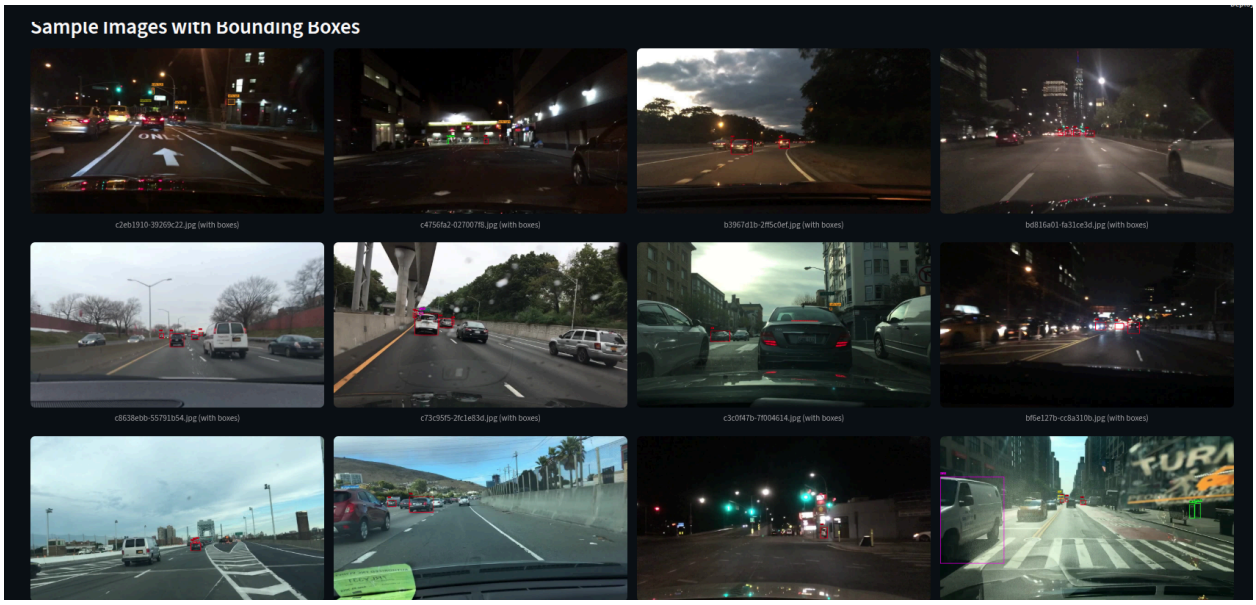
Sample of missing detections

Correct Predictions



The model demonstrates good generalization in correctly predicting classes. This is supported by the convergence of the classification loss and by visual results, which show consistent performance across different conditions, such as daytime, nighttime, and varied scenes.

Missed Predictions



Some cases reveal clear shortcomings. For example:

- Missing rate is significantly high at night, as the number of objects missed at night is much more than day compared to daytime. This indicates that low-light environments pose challenges for detection.
- Class ambiguity also reduces performance. The label *truck* is used interchangeably for ambulances, garbage trucks, and vans, even though these are not distinct classes. Such overlap confuses the model and leads to incorrect detections.

Small Object Performance

- **Correct detections** (avg. width: 59.2 px, height: 54.0 px, area: 7,735 px²).

- **Missed detections** (avg. width: 69.5 px, height: 60.6 px, area: 10,146 px²).

While many small annotations are captured successfully, incorrect predictions often occur for objects that are either too small to detect reliably or too large, such as trucks or vans occupying most of the image. This effect is particularly visible at the sides of vehicles. By contrast, smaller cars are detected more consistently, likely because they dominate the dataset distribution.

Key Insights

1. **Generalization is decent** – classification works well across varied conditions.
2. **Dark environments reduce recall**, making nighttime detection more difficult.
3. **Class ambiguity (truck/van/ambulance)** significantly affects performance.
4. **Very small objects and huge objects remain the hardest to detect**. This is clear from the examples, that there is a certain range of bounding box areas within which a bounding box is easiest to detect.

Future

- Train for more epochs to reach convergence.
- Try using simulation data for more robustness.
- Use learning rate warmup and scheduling to stabilize early training.
- Clearly define unique classes
- Apply stronger augmentations (mosaic, brightness/saturation, shifts, noise, random crop.