

Evaluation

Detection

Submission format

To evaluate your algorithms on the BDD100K detection benchmark, you may prepare your prediction results using the [Scalabel Format](#)[↗]. Specifically, these fields are required:

```
- name: str, name of current frame
- labels []:
  - id: str, not used here, but required for loading
  - category: str, name of the predicted category
  - score: float
  - box2d []:
    - x1: float
    - y1: float
    - x2: float
    - y2: float
```

You can submit your predictions to our [evaluation server](#)[↗] hosted on EvalAI. The submission file needs to be a JSON file.

Run Evaluation on Your Own

You can evaluate your algorithm with public annotations by running:

```
python3 -m bdd100k.eval.run -t det -g ${gt_file} -r ${res_file}
```

- `gt_file`: ground truth file in JSON in Scalabel format.
- `res_file`: prediction results file in JSON, format as described above.

Other options. - You can specify the output file to save the evaluation results to by adding `-out-file ${out_file}`.

Evaluation Metrics

Similar to COCO evaluation, we report 12 scores as “AP”, “AP_50”, “AP_75”, “AP_small”, “AP_medium”, “AP_large”, “AR_max_1”, “AR_max_10”, “AR_max_100”, “AR_small”, “AR_medium”, “AR_large” across all the classes.

Instance Segmentation

We use the same metrics set as for detection above. The only difference lies in the computation of distance matrices. Concretely, in detection, it is computed using box IoU. While for instance segmentation, mask IoU is used.

Submission format

To evaluate your algorithms on the BDD100K instance segmentation benchmark, you may prepare predictions in RLE format (consistent with COCO format) or bitmask format (illustrated in [Instance Segmentation Bitmask](#)). For RLE, these fields are required:

```
- name: str, name of current frame
- labels []:
  - id: str, not used here, but required for loading
  - category: str, name of the predicted category
  - score: float
  - rle:
    - counts: str
    - size: (height, width)
```

For bitmask, in addition to the folder of bitmask images, a JSON file is needed, with the following format:

```
- name: str, name of the input image,
- labels []:
  - id: str, not used here, but required for loading
  - index: int, in range [1, 65535], the index in B and A channel
  - score: float, confidence score of the prediction
```

- *index*: the value corresponds to the “ann_id” stored in B and A channels.

You can submit your predictions to our [evaluation server](#)[↗] hosted on EvalAI. Currently, the evaluation server only supports bitmask format.

Run Evaluation on Your Own

You can evaluate your algorithm with public annotations by running:

```
python3 -m bdd100k.eval.run -t ins_seg -g ${gt_path} -r ${res_path} --score-file  
${res_score_file}
```

- *gt_path*: the path to the ground-truth JSON file or bitmasks images folder.
- *res_path*: the path to the results JSON file or bitmasks images folder.
- *res_score_file*: the JSON file with the confidence scores (for bitmasks).

Other options. - You can specify the output file to save the evaluation results to by adding *-out-file \${out_file}*.

Note that the evaluation results for RLE format and bitmask format are not exactly the same, but the difference is negligible (< 0.1%).

Pose Estimation

We use the same metrics set as detection and instance segmentation above. The only difference lies in the computation of distance matrices. Concretely, in detection, it is computed using box IoU. While for instance segmentation, mask IoU is used. For pose estimation, object keypoint similarity is used (OKS).

Submission format

To evaluate your algorithms on the BDD100K pose estimation benchmark, you may prepare your prediction results using the [Scalabel Format](#)[↗]. Specifically, these fields are required:

```
- name: str
- labels []:
  - id: str, not used here, but required for loading
  - category: "pedestrian"
  - score: float
- graph:
  - nodes []:
    - location: (int, int), (x, y) position of node
    - category: str, joint name
    - id: int, unique id for node used for defining edges
    - score: float
  - edges []:
    - source: str, source node id
    - target: str, target node id
    - type: str, type of edge
  - type: "Pose2D-18Joints_Pred"
```

Run Evaluation on Your Own

You can evaluate your algorithm with public annotations by running:

```
python3 -m bdd100k.eval.run -t pose -g ${gt_file} -r ${res_file}
```

- `gt_file`: ground truth file in JSON in Scalabel format.
- `res_file`: prediction results file in JSON, format as described above.

Other options. - You can specify the output file to save the evaluation results to by adding `-out-file ${out_file}`.

Evaluation Metrics


Similar to COCO evaluation, we report 10 scores as “AP”, “AP_50”, “AP_75”, “AP_medium”, “AP_large”, “AR”, “AR_50”, “AR_75”, “AR_medium”, “AR_large” across all the classes.

Panoptic Segmentation

We use the same metrics as COCO panoptic segmentation. PQ, RQ and SQ are computed for things, stuffs, and all categories.

Submission format

To evaluate your algorithms on the BDD100K panoptic segmentation benchmark, you may prepare predictions in RLE or bitmask format (illustrated in [Panoptic Segmentation Bitmask](#)). See [Instance Segmentation Evaluation](#) for the RLE format.

[1] Kirillov, A., He, K., Girshick, R., Rother, C., & Dollár, P. (2019). Panoptic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 9404-9413). 

Run Evaluation on Your Own

You can evaluate your algorithm with public annotations by running:

```
python3 -m bdd100k.eval.run -t pan_seg -g ${gt_path} -r ${res_path}
```

- *gt_path*: the path to the ground-truth JSON file or bitmasks images folder.
- *res_path*: the path to the results JSON file or bitmasks images folder.

Other options. - You can specify the output file to save the evaluation results to by adding *-out-file \${out_file}*.

Note that the evaluation results for RLE format and bitmask format are not exactly the same, but the difference is negligible (< 0.1%).

Semantic Segmentation

We assess the performance using the standard Jaccard Index, commonly known as mean-IoU. Moreover, IoU for each class are also displayed for reference.

Submission format

To evaluate your algorithms on the BDD100K semantic segmentation benchmark, you may prepare predictions in RLE or mask format. For RLE, these fields are required:

```
- name: str, name of current frame
- labels []:
  - id: str, not used here, but required for loading
  - category: str, name of the predicted category
  - rle:
    - counts: str
    - size: (height, width)
```

For masks, the submission should be a folder of masks.

You can submit your predictions to our [evaluation server](#)[↗] hosted on EvalAI. Currently, the evaluation server only supports mask format.

Run Evaluation on Your Own

You can evaluate your algorithm with public annotations by running:

```
python3 -m bdd100k.eval.run -t sem_seg -g ${gt_path} -r ${res_path}
```

- *gt_path*: the path to the ground-truth JSON file or masks images folder.
- *res_path*: the path to the results JSON file or masks images folder.

Other options. - You can specify the output file to save the evaluation results to by adding *-out-file \${out_file}*.

Note that the evaluation results for RLE format and mask format are not exactly the same, but the difference is negligible (< 0.1%).

Drivable Area

The drivable area task applies the same rule with semantic segmentation. One notable difference is that they have different class definitions and numbers. Another is that the prediction of background pixels matters for drivable area. Unlike semantic segmentation, which ignores *unknown* pixels, drivable area instead takes consideration of *background* pixels when computing IoUs. Though the *background* class is not counted into the final mIoU.

Submission

You can submit your predictions to our [evaluation server](#)[↗] hosted on EvalAI. Currently, the evaluation server only supports mask format.

Run Evaluation on Your Own

You can evaluate your algorithm with public annotations by running:

```
python3 -m bdd100k.eval.run -t drivable -g ${gt_path} -r ${res_path}
```

- *gt_path*: the path to the ground-truth JSON file or masks images folder.
- *res_path*: the path to the results JSON file or masks images folder.

Other options. - You can specify the output file to save the evaluation results to by adding *-out-file \${out_file}*.

Note that the evaluation results for RLE format and bitmask format are not exactly the same, but the difference is negligible (< 0.1%).

Lane Marking

The lane marking takes the F-score [1] as the measurement. We evaluate the F-score for each category of the three sub-tasks with threshold as 1, 2 and 5 pixels. Before the evaluation, morphological thinning is adopted to get predictions of 1-pixel width. For each sub-task, the mean F-score will be showed. The main item for the leaderboard is the averaged mean F-score of these three sub-tasks.

[1] [A Benchmark Dataset and Evaluation Methodology for Video Object Segmentation](#). F. Perazzi, J. Pont-Tuset, B. McWilliams, L. Van Gool, M. Gross, and A. Sorkine-Hornung. Computer Vision and Pattern Recognition (CVPR) 2016[↗]

Submission format

To evaluate your algorithms on the BDD100K lane marking benchmark, you may prepare predictions in RLE format or mask format (illustrated in [Lane Marking Format](#)).

Run Evaluation on Your Own

You can evaluate your algorithm with public annotations by running:

```
python3 -m bdd100k.eval.run -t lane_mark -g ${gt_path} -r ${res_path}
```

- *gt_path*: the path to the ground-truth JSON file or masks images folder.
- *res_path*: the path to the results JSON file or masks images folder.

Other options. - You can specify the output file to save the evaluation results to by adding *-out-file \${out_file}*.

Multiple Object Tracking

Submission format

To evaluate your algorithms on BDD100K multiple object tracking benchmark, the submission must be in one of these formats:

- A zip file of a folder that contains JSON files of each video.
- A zip file of a file that contains a JSON file of the entire evaluation set.

The JSON file for each video should contain a list of per-frame result dictionaries with the following structure:

```
- videoName: str, name of current sequence
- name: str, name of current frame
- frameIndex: int, index of current frame within sequence
- labels []:
  - id: str, unique instance id of prediction in current sequence
  - category: str, name of the predicted category
  - box2d []:
    - x1: float
    - y1: float
    - x2: float
    - y2: float
```

You can find an example result file in [bdd100k.eval.testcases](#) 

You can submit your predictions to our [evaluation server](#)  hosted on EvalAI.

Run Evaluation on Your Own

You can evaluate your algorithms with public annotations by running:

```
python -m bdd100k.eval.run -t box_track -g ${gt_file} -r ${res_file}
```

Other options. - You can specify the output file to save the evaluation results to by adding `-out-file ${out_file}`.

Evaluation Metrics

We employ mean Multiple Object Tracking Accuracy (mMOTA, mean of MOTA of the 8 categories) as our primary evaluation metric for ranking. We also employ mean ID F1 score (mIDF1) to highlight the performance of tracking consistency that is crucial for object tracking. All metrics are detailed below. Note that the overall performance is measured for all objects without considering the category if not mentioned.

- mMOTA (%): mean Multiple Object Tracking Accuracy across all 8 categories.
- mIDF1 (%): mean ID F1 score across all 8 categories.
- mMOTP (%): mean Multiple Object Tracking Precision across all 8 categories.
- MOTA (%): Multiple Object Tracking Accuracy [1]. It measures the errors from false positives, false negatives and identity switches.
- IDF1 (%): ID F1 score [2]. The ratio of correctly identified detections over the average number of ground-truths and detections.
- MOTP (%): Multiple Object Tracking Precision [1]. It measures the misalignments between ground-truths and detections.
- FP: Number of False Positives [1].
- FN: Number of False Negatives [1].
- IDSw: Number of Identity Switches [1]. An identity switch is counted when a ground-truth object is matched with a identity that is different from the last known assigned identity.
- MT: Number of Mostly Tracked identities. At least 80 percent of their lifespan are tracked.
- PT: Number of Partially Tracked identities. At least 20 percent and less than 80 percent of their lifespan are tracked.
- ML: Number of Mostly Lost identities. Less of 20 percent of their lifespan are tracked.

- FM: Number of FragMentations. Total number of switches from tracked to not tracked detections.

[1] Bernardin, Keni, and Rainer Stiefelhagen. "Evaluating multiple object tracking performance: the CLEAR MOT metrics." EURASIP Journal on Image and Video Processing 2008 (2008): 1-10. [↗](#)

[2] Ristani, Ergys, et al. "Performance measures and a data set for multi-target, multi-camera tracking." European Conference on Computer Vision. Springer, Cham, 2016. [↗](#)

Super-category

In addition to the evaluation of all 8 classes, we also evaluate results for 3 super-categories specified below. The super-category evaluation results are provided only for the purpose of reference.

```
"HUMAN": ["pedestrian", "rider"],  
"VEHICLE": ["car", "bus", "truck", "train"],  
"BIKE": ["motorcycle", "bicycle"]
```

Ignore regions

After the bounding box matching process in evaluation, we ignore all detected false-positive boxes that have >50% overlap with the crowd region (ground-truth boxes with the "Crowd" attribute).

We also ignore object regions that are annotated as 3 distracting classes ("other person", "trailer", and "other vehicle") by the same strategy of crowd regions for simplicity.

Pre-training

It is a fair game to pre-train your network with **ImageNet**, but if other datasets are used, please note in the submission description. We will rank the methods without using external datasets except **ImageNet**.

Multi Object Tracking and Segmentation (Segmentation Tracking)

We use the same metrics set as MOT above. The only difference lies in the computation of distance matrices. Concretely, in MOT, it is computed using box IoU. While for MOTS, mask IoU is used.

Submission format

The submission should be in the same format as for MOT with RLE. Additionally, it can also be a zipped nested folder for bitmask images, where images belonging to the same video are placed in the same folder, named by `${videoName}`.

For RLE, the JSON file for each video should contain a list of per-frame result dictionaries with the following structure:

```
- videoName: str, name of current sequence
- name: str, name of current frame
- frameIndex: int, index of current frame within sequence
- labels []:
  - id: str, unique instance id of prediction in current sequence
  - category: str, name of the predicted category
  - rle:
    - counts: str
    - size: (height, width)
```

You can find an example file [here](#).

You can submit your predictions to our [evaluation server](#) hosted on EvalAI. Currently, the evaluation server only supports bitmask format.

Run Evaluation on Your Own

You can evaluate your algorithms with public annotations by running:

```
python -m bdd100k.eval.run -t seg_track -g ${gt_path} -r ${res_path}
```

- `gt_path`: the path to the ground-truth JSON file or bitmasks images folder.

- *res_path*: the path to the results JSON file or bitmasks images folder.

Other options. - You can specify the output file to save the evaluation results to by adding *-out-file \${out_file}*.

Note that the evaluation results for RLE format and bitmask format are not exactly the same, but the difference is negligible (< 0.1%).