


main 1 branch 1 tag Go to file Add file Code

 cy-xu update readme and help 4e2b068 22 days ago 54 commits

boi_dataset	clean up old files	26 days ago
media	add API demo	last month
yolov3	update readme and help	22 days ago
.gitattributes	add trained model to lfs	25 days ago
.gitignore	improved labeling stats, expanded dataset	last month
Dockerfile	Dockerfile improve	26 days ago
README.md	update readme and help	22 days ago
boi_model_API_demo.py	Use original ubuntu image and miniconda install	last month

About Computer Vision Modeling for Baltimore Trash Wheel Images Readme

Releases 1 v0.1.0 initial release Latest 25 days ago

Packages No packages published Publish your first package

Contributors 4 cy-xu CY Xu crlang44 Chris Lang aaronroan Aaron Roan OlivelliAri Arianna Olivelli

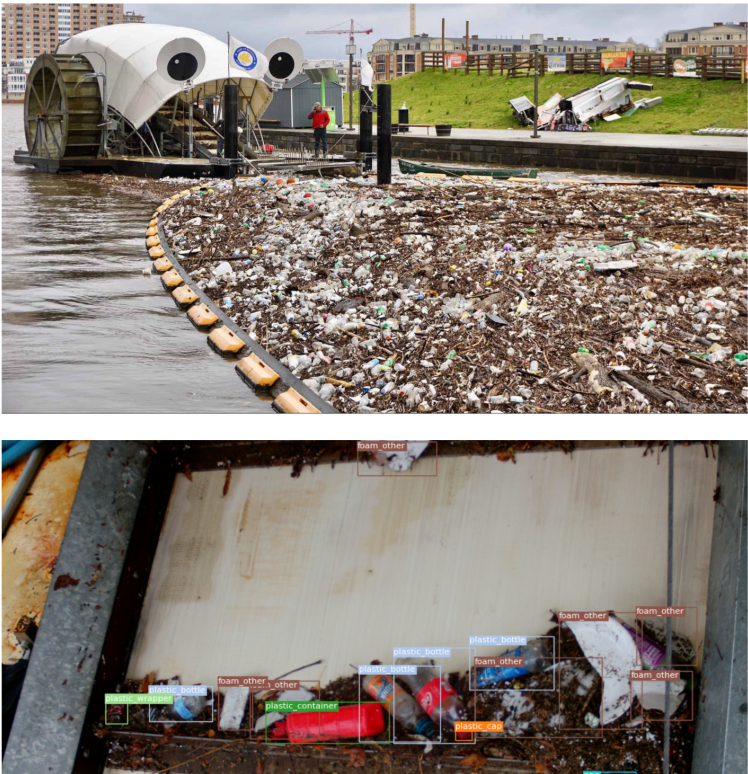
Languages Python 89.2% Shell 9.2% Dockerfile 1.6%

# BOI Baltimore Trash Wheel Computer Vision Model and Dataset

The Baltimore Trash Wheel Computer Vision Model was developed by researchers at the [Benioff Ocean Initiative](#) (BOI) at UC Santa Barbara. The model uses deep-learning to support operators of the [Baltimore Trash Wheels](#) in identifying ocean-bound river waste to collect data more efficiently and more accurately.

We produced a custom dataset, [BOI Baltimore Trash Wheel Dataset](#), which was used to train a PyTorch-based [YOLOv3](#) model to detect 15 different classes of river wastes like plastic bottles or bags, foam containers or fragments, and other inorganic wastes in images taken from cameras mounted on the trash wheels in Baltimore.

The developers' intent for this model and dataset is for their use by other practitioners and researchers working on capturing and studying waste in rivers to support a greater understanding of the types and sources of river waste and to ultimately turn off the tap of plastic and other solid waste pollution into the ocean.





## Usage via Docker

- install the docker application: <https://hub.docker.com/>
- pull the detector image we built from "docker hub" (pending formal release)

## Build new docker image

```
# build the docker image, name it cccml
docker build . -f Dockerfile -t cccml
```

- after building the image locally, it's ready to run in the container

```
# start docker and enter the detector application
# -it: interactive mode
# -v: mapping current directory to /data in Docker
# --shm-size: increase shared memory to 4GB
docker run -it -v "$(pwd)":/data --shm-size=4gb cccml /bin/bash

# assuming all target images are saved in the `images` directory, the dection results and the report

# inside the docker, run the same detection commands explained below
poetry run yolo-detect --conf_thres 0.1 --images /data/images --output /data/detection_result

# accessing GPUs
https://docs.docker.com/engine/reference/commandline/run/#access-an-nvidia-gpu
```

## Manual installation

```
git clone https://github.com/Benioff0ceanInitiative/cccBaltimoreTrashWheelML
cd cccBaltimoreTrashWheelML/yolov3

# create a Python virtual environment named cccml for this project
conda create -n cccml python=3.8 -y
conda activate cccml

# install the dependent packages using poetry
pip3 install poetry --user
poetry install
```

## Download the trained model

```
# the trained model is now included and saved as
yolov3/weights/boi_baltimore_model.pth

# download yolo pretrained models (for training new models via transfer learning)
bash download_weights.sh
```

## Detect objects, visualize results, and produce reports

```
# detection, training, and evaluation should take palce inside the yolov3 directory
cd cccBaltimoreTrashWheelML/yolov3

poetry run yolo-detect \
--images boi_dataset/valid.txt \
--conf_thres 0.1 \
--nms_thres 0.5 \
--img_size 1024 \
--start_date 2019-02-01 \
--end_date 2022-03-01 \
--weights weights/boi_baltimore_model.pth \
--output ./results_output
```

`--images` could be a directory including many images, or a `.txt` file with file paths pointing to images

`--start_date` and `--end_date` help filter the data taken within a specific time duration

The BOI model is trained on `1024x1024` images, the best prediction results is expected at the same resolution.

Lower the confidence threshold `conf_thres` to include more detections (at the risk of higher false positive rate), it is safe to use a lower `conf_thres` of 0.4 with a higher `nms_thres` of 0.5 as the distracted drivers are represented by

is safe to use a lower `conf_thres` or 0.1 with a higher `nms_thres` or 0.5 as the duplicated boxes are removed by the higher threshold, leaving the most fitting bounding box.

A detection report is saved in `.txt` and `.csv` formats for future interpretation (location is parsed from file names so the target files should follow the previously defined naming format):

Index	Class	Harris	Jones	Mason	Total
0	plastic_bag	10	1	0	11
1	plastic_bottle	176	44	0	220
2	plastic_cap	54	28	1	83
etc ...					

## Train a new model

### Images and annotations

Before training for the first time, please follow the [dataset readme](#) to prepare the data.

By default, download the images and place them in `boi_dataset/images/`. The CVAT annotation files should be placed in `boi_dataset/annotations`. The `boi_dataset/label_converter.py` converts the CVAT annotation to the YOLO format needed for training new models.

The training scripts access the data through a symbolic link at `yolo3/boi_dataset/`, which is pointed to `boi_dataset/`.

annotations to `data/custom/labels/`. The dataloader expects that the annotation file corresponding to the image `data/custom/images/train.jpg` has the path `data/custom/labels/train.txt`. Each row in the annotation file should define one bounding box, using the syntax `label_idx x_center y_center width height`. The coordinates should be scaled `[0, 1]`, and the `label_idx` should be zero-indexed and correspond to the row number of the class name in `data/custom/classes.names`.

If the number of classes are changed, following the [instruction from Yolo](#) to modify the neural network config file accordingly.

```
cd cccBaltimoreTrashWheelML/yolo3

poetry run yolo-train \
--pretrained_weights weights/darknet53.conv.74 \
--epochs 5000 \
--multiscale_training \
--seed 0 \
--comment multiscale_64-8-1024_1k24k27k
```

Load a Darknet-53 backend weights pretrained on ImageNet `darknet53.conv.74` to gain a better start through transfer learning.

The `multiscale_training` flag allows more generalized detection.

### Track training progress in Tensorboard

- Initialize training
- Run `tensorboard --logdir='checkpoints'` in the console to start tensorboard server
- Track training progress at <http://localhost:6006/>

## Evaluate model performance

```
cd cccBaltimoreTrashWheelML/yolo3

poetry run yolo-test \
--iou_thres 0.4 \
--nms_thres 0.5 \
--conf_thres 0.01 \
--img_size 1024 \
--weights checkpoints/training_dir/target_checkpoint.pth
```

A model performance report is generated after evaluation (limited by the small sample size of certain classes, model performance on class 1, 2, 4, 5, 7 are statistically meaningful at the moment):

Index	Class	AP	Objects
0	plastic_bag	0.35333	20
1	plastic_bottle	0.85126	219

2	plastic_cap	0.65524	103
3	plastic_container	0.10000	20
4	plastic_wrapper	0.63467	85
5	plastic_other	0.35607	162
6	foam_container	0.18182	11
7	foam_other	0.86932	274
8	glass_bottle	0.00000	5
9	paper_container	0.25000	4
10	paper_other	0.25000	4
11	metal_cap	0.00000	3
12	metal_can	0.00000	4
13	ppe	0.25000	4
14	misc	0.04167	60

## API

An example prediction call from a simple OpenCV python script would look like this:

```
# for model inferencing (detection) only
import cv2
from yolov3.pytorchyolo import detect, models

# configuration file that define the nernal network
model_config = './boi_dataset/yolov3_boi.cfg'

# path to the trained model weights
trained_model = './yolov3/weights/boi_baltimore_model.pth'

# load the trained weights to the model
model = models.load_model(model_config, trained_model)

# load a sample image, convert OpenCV bgr to rgb
image = cv2.imread('./media/wp_b_harris creek_20201100_10300028_raw.JPG')

# Runs the detection model on the image
boxes = detect.detect_image(model, image, img_size=1024, conf_thres=0.1, nms_thres=0.5)

print(boxes)
# Output will be a numpy array in the following format:
# [[x1, y1, x2, y2, confidence, class]]
```

## BOI Baltimore Trash Wheel Dataset

Index	Class	Objects
0	plastic_bag	107
1	plastic_bottle	1167
2	plastic_cap	566
3	plastic_container	89
4	plastic_wrapper	543
5	plastic_other	919
6	foam_container	87
7	foam_other	1601
8	glass_bottle	17
9	paper_container	15
10	paper_other	13
11	metal_cap	8
12	metal_can	23
13	ppe	29
14	misc	267



## Credit

python 3.7 | 3.8 | 3.9  PyPI license (license pending discussion)

### Benioff Ocean Initiative (BOI)

<https://boi.ucsb.edu/>

### Clean Currents Coalition

<https://cleancurrentscoalition.org/>

### MR. TRASH WHEEL® : A PROVEN SOLUTION TO OCEAN PLASTICS

<https://www.mrtrashwheel.com/>

### PyTorch-YOLOv3

This project is built upon [Erik's YOLOv3 implementation](#) in PyTorch.

### YOLOv3: An Incremental Improvement

*Joseph Redmon, Ali Farhadi*

#### Abstract

We present some updates to YOLO! We made a bunch of little design changes to make it better. We also trained this new network that's pretty swell. It's a little bigger than last time but more accurate. It's still fast though, don't worry. At  $320 \times 320$  YOLOv3 runs in 22 ms at 28.2 mAP, as accurate as SSD but three times faster. When we look at the old .5 IOU mAP detection metric YOLOv3 is quite good. It achieves 57.9 AP50 in 51 ms on a Titan X, compared to 57.5 AP50 in 198 ms by RetinaNet, similar performance but 3.8× faster. As always, all the code is online at <https://pjreddie.com/yolo/>.

[\[Paper\]](#) [\[Project Webpage\]](#) [\[Authors' Implementation\]](#)

```
@article{yolov3,
  title={YOLOv3: An Incremental Improvement},
  author={Redmon, Joseph and Farhadi, Ali},
  journal = {arXiv},
  year={2018}
}
```

