

Milestone 3 (Final)
Zaka AIC C3 - Capstone Project



Waste Detection in the Wild using mobile phones

By Oreyon



Abdulrahim El Mohamad
Nabil Miri

Problem definition

This project deals with the problem of littering the environment. The aim is to build a litter monitoring system that can perform trash image segmentation and type detection. Image segmentation is the idea of assigning a label to pixels in images and it is divided into 2 types based on whether we need to differentiate between the object instances in the image or not. Semantic segmentation treats multiple objects as one category whereas Instance segmentation identifies individual objects within the pre-known categories.

The massive creation of disposable goods over the past few years has significantly increased the amount of waste generated which most of it is thrown in the wild. In order to assist the collection of trash, ML-based solution could be used. This will help prevent additional environmental degradation and, as a result, protect the existence of humans and other wild organisms.

How the problem has been previously solved

1) [Litter Detection with Deep Learning: A Comparative Study](#): The experiments considered two datasets, namely, PlastOPol and TACO, and aimed to assess the performance of the detectors using mobile devices. The experimental results showed that YOLO-v5x outperformed the other state-of-the-art methods in both datasets. Moreover, YOLO-v5s proved to be the most promising approach to be run in mobile devices due to its competitive results and its ability to process up to 5.19 frames per second (FPS) in a commercial smartphone.

Table 6. Litter detection results on TACO (best results appear in bold).

Methods	AP50	AP@	AR@	F1@
RetinaNet [41]	50.6	26.7	37.1	31.1
Faster R-CNN [28]	51.1	28.1	36.9	31.9
Mask R-CNN [43]	52.3	29.2	38.6	33.2
EfficientDet-d0 [30]	32.7	23.8	28.4	25.9
EfficientDet-d5 [30]	42.3	35.2	40.3	37.6
YOLO-v5s [32]	54.7	38.8	58.1	46.5
YOLO-v5x [32]	63.3	48.4	66.4	56.0

But no segmentation was performed only based on object detection.

2) [Smart street litter detection and classification based on Faster R-CNN and edge computing](#): Another paper developed a RCNN model that was able to detect litter in streets using dataset they made using google maps street images, ImageNet images, google search images and other collected by them. There are approximately 7059 photos which were manually labelled by them. These were divided into 11 labels. Also in this paper, no segmentation was done.

3) [A Multi-Level Approach to Waste Object Segmentation](#): In this paper they used a multi level approach to waste object localization. Using the TACO dataset deeplabv3 performed well and better than FCN-8s. but when adding their proposed architecture to these models the metrics becomes better. Moreover, they introduced their dataset MJU which is an indoor dataset consisting of RGBD images.

Dataset: TACO (Test)					
	Backbone	IoU	mIoU	Prec	Mean
Baseline Approaches					
FCN-8s [17]	VGG-16	70.43	84.31	85.50	92.21
DeepLabv3 [23]	ResNet-101	83.02	90.99	88.37	94.00
Proposed Multi-Level (ML) Model					
FCN-8s-ML	VGG-16	74.21 (+3.78)	86.35 (+2.04)	90.36 (+4.86)	94.65 (+2.44)
DeepLabv3-ML	ResNet-101	86.58 (+3.56)	92.90 (+1.91)	92.52 (+4.15)	96.07 (+2.07)

Here they used TACO-1 where all classes were combined into 1 class only called 'litter'. Segmentation was performed.

4) [Instance Segmentation of Multiclass Litter and Imbalanced Dataset Handling – A Deep Learning Model](#)

Comparison: We can see that in this paper they used TACO dataset and tested Mask R-CNN and DetectroRS for instance segmentation. DetectroRS performed better but it demands more GPU than Mask R-CNN. They used these models for instance segmentation. Moreover, they tested the effect of data oversampling on improving the metrics.

Metric	area	Mask R-CNN score	DetectroRS score	diff
mAP	all	0.127	0.167	+0.040
mAP _{0.5}	all	0.159	0.203	+0.044
mAP _{0.75}	all	0.136	0.178	+0.042
mAP	small	0.022	0.044	+0.022
mAP	medium	0.064	0.140	+0.076
mAP	large	0.150	0.194	+0.044
mAR	all	0.320	0.479	+0.159
mAR	small	0.021	0.058	+0.037
mAR	medium	0.093	0.252	+0.159
mAR	large	0.367	0.507	+0.140

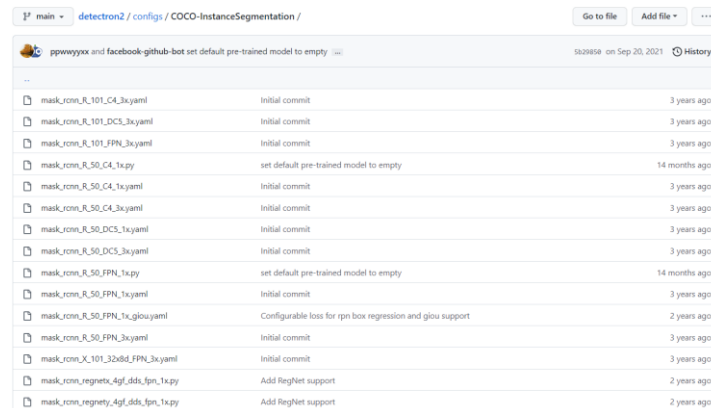
5) Pedro F Proença and Pedro Simões. “TACO: Trash Annotations in Context for Litter Detection: They used this dataset with Mask R-CNN with a random split of proportions 80%, 10%, 10% for train, validation, and test respectively with 4-fold cross-validation and got a mAP50 of 19.4 ~ 1.5. They also reported results of their single class dataset TACO-1 using the split method, which got 26.2 ~ 1.0 AP50. Resnet-50 was used as backbone, and images were resized and padded to 1024 times 1024 pixels.

Final proposed approach (with technical details)

Model Used:

For the model we used Detectron2 which is Facebook's new vision library that allows us to easily use and create object detection, instance segmentation, keypoint detection, and panoptic segmentation models. In addition, it has a simple, modular design that makes it easy to rewrite a script for another dataset. There are several built-in models that we can choose from.

The full list of available instance segmentation models can be found in the below tables:



The screenshot shows the GitHub repository for Detectron2, specifically the 'configs/COCO-InstanceSegmentation' directory. It lists various pre-trained models and their commit dates. The models are organized into a table with columns for the model name, the commit message, and the time since the commit.

Model Name	Commit Message	Time
mask_rcnn_R_101_C4_3x.yaml	Initial commit	3 years ago
mask_rcnn_R_101_DC5_3x.yaml	Initial commit	3 years ago
mask_rcnn_R_101_FPN_3x.yaml	Initial commit	3 years ago
mask_rcnn_R_50_C4_1x.py	set default pre-trained model to empty	14 months ago
mask_rcnn_R_50_C4_1x.yaml	Initial commit	3 years ago
mask_rcnn_R_50_C4_3x.yaml	Initial commit	3 years ago
mask_rcnn_R_50_DC5_1x.yaml	Initial commit	3 years ago
mask_rcnn_R_50_DC5_3x.yaml	Initial commit	3 years ago
mask_rcnn_R_50_FPN_1x.py	set default pre-trained model to empty	14 months ago
mask_rcnn_R_50_FPN_1x.yaml	Initial commit	3 years ago
mask_rcnn_R_50_FPN_1x_giou.yaml	Configurable loss for rpn box regression and giou support	2 years ago
mask_rcnn_R_50_FPN_3x.yaml	Initial commit	3 years ago
mask_rcnn_X_101_32x8d_FPN_3x.yaml	Initial commit	3 years ago
mask_rcnn_regnetx_4g_dds_fpn_1x.py	Add RegNet support	2 years ago
mask_rcnn_regnetx_4g_dds_fpn_1x.py	Add RegNet support	2 years ago

Dataset Used:

Currently, there exist some litter datasets; however, most of them were built in controlled setups, i.e., with only one instance of litter per image or taken in indoor scenarios for recycling. Approaches developed based on such image collections cannot be generalized for real-world scenarios.

There is a dataset called “PlastOPol,” which is based on images taken through the Marine Debris Tracker, a set of 2418 images with the presence of litter in a realistic context covering several types of environments, i.e., urban, beaches, forests and flint fields, and including different types of litter,

including plastic, glass, metal, paper, cloth and rubber, among others. The problem is that this dataset doesn't have segmented annotations just BBox.

There is also UUAVWaste dataset that contains segmentation images for trash taken by drone.

MJU, is another dataset that contains RGBD litter images that were taken in a lab and has annotations for segmentations. The problem with this dataset is that the images are taken indoor from a lab where a person is holding the waste unlike found in reality.

TACO is the only publicly available dataset containing 1500 images from realistic outdoor scenario and with annotations for segmentation. Thus, we used this dataset.

Some comparison with other datasets:

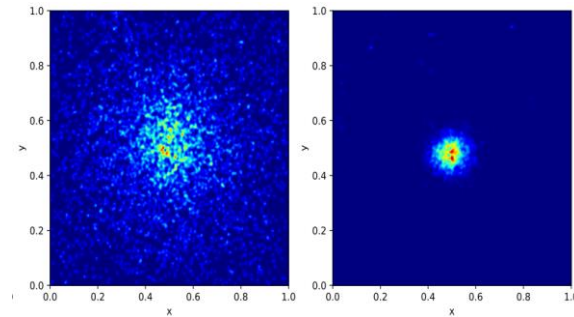
Based on BBox area:

Table 2. Datasets employed in the comparative study.

Dataset	# Images	# Bounding Boxes by Area			# Annotations
		Small ¹	Medium ²	Large ³	
TACO [37]	1500	384	1305	3095	4784
PlastOpol	2418	33	445	4822	5300

¹ Small \rightarrow area $\leq 32^2$. ² Medium $\rightarrow 32^2 <$ area $\leq 96^2$. ³ Large \rightarrow area $> 96^2$.

Based on litter location in the image:



The first is for the TACO dataset and the latter is for the MJU.

TACO Dataset:

TACO (<http://tacodataset.org/stats>) has 1500 images with 4784 annotations (avg 3.19 object per images). This dataset labeled its images under 60 categories corresponding to 28 super categories; however, an experimental one-category set-up was also proposed by the authors where all the litter instances belong to the "litter" class. It is open-source, and anyone could help in labelling. The new labels are added to the unofficial dataset whereas only after the authors review of the annotation, they would be added to the main dataset. TACO is a very challenging dataset due to the presence of bottles, bottle caps, glass, rope, strings, etc. Additionally, there is many samples from the cigarette class in the ground truth. This type of litter is covered by bounding boxes with a size less than 64×64 pixels, which makes it a very complex scenario for detection.

Sample of the labelled and segmented images:



The format of the annotations is provided in COCO format.

EDA:

COCO Format:

```
dataset.keys()

dict_keys(['info', 'images', 'annotations', 'scene_annotations', 'licenses', 'categories', 'scene_categories'])
```

Info: here we have some info about when the dataset was created and some other descriptions

```
dataset['info'].keys()

dict_keys(['year', 'version', 'description', 'contributor', 'url', 'date_created'])
```

Images:

```
dataset['images'][1]

{'id': 1,
 'width': 1537,
 'height': 2849,
 'file_name': 'batch_1/000008.jpg',
 'license': None,
 'flickr_url': 'https://farm66.staticflickr.com/65535/47803331152_ee00755a2e_o.png',
 'coco_url': None,
 'date_captured': None,
 'flickr_640_url': 'https://farm66.staticflickr.com/65535/47803331152_10bean025a_z.jpg'}
```

It contains image id, size, file name (or location) and a url where it could be found online.

Annotations:

The annotations contains the annotation id, image id to which the annotation belongs to (which is 'id' in the image list), category id (such as metals, plastics and so on), area, segmentation and bbox annotations values.

For the bounding box usually there is a bbox_mode which is not present here as we are using the default configuration. We have multiple choices [x_0, y_0, x_1, y_1], [x_0, y_0, width, height] or any other representation. However, the bbox_mode should be consistent with bbox representation. Moreover, Iscrowd = 0 (means it's a single object).

Segmentation is a polygon with n points, (x_i, y_i) 'segmentation': [[x_0, y_0, x_1, y_1, ..., x_n, y_n]], Default is BoxMode.XYXY_ABS (x0, y0, x1, y1) in absolute floating points coordinates. The coordinates in range [0, width or height].

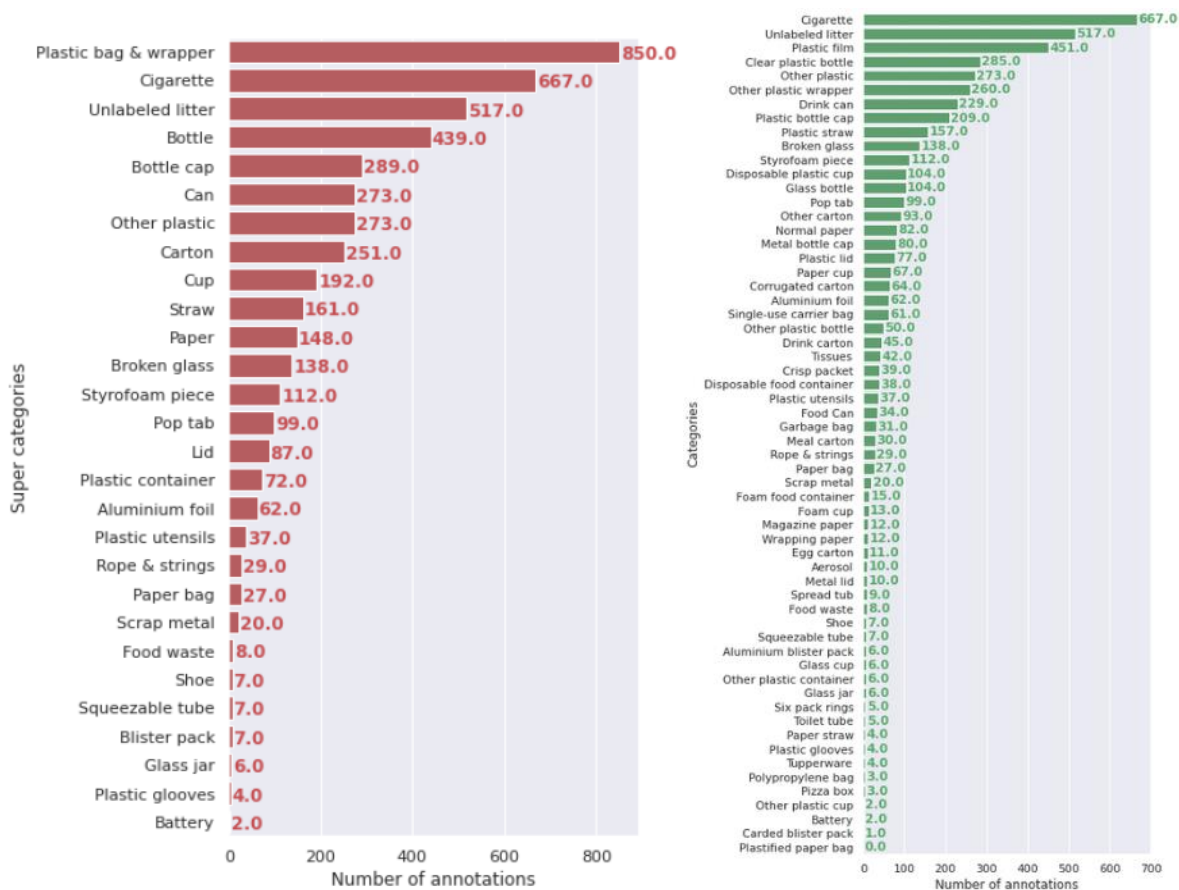
```
dataset['annotations'][7]

{'id': 8,
 'image_id': 3,
 'category_id': 7,
 'segmentation': [[643.0,
 1453.0,
 649.0,
 1445.0,
 653.0,
 1442.0,
 657.0,
 1450.0,
 663.0,
 1459.0,
 662.0,
 1467.0,
 656.0,
 1478.0,
 651.0,
 1481.0,
 644.0,
 1476.0,
 643.0,
 1465.0,
 638.0,
 1459.0,
 634.0,
 1459.0,
 643.0,
 1453.0]],
 'area': 578.5,
 'bbox': [634.0, 1442.0, 29.0, 39.0],
 'iscrowd': 0}
```

Scene annotation: depicts the background such as clean, indoor, pavement, vegetation, water, etc.

Categories:

The below charts depict the number of annotations in the 60 categories and the 28 super categories.



There are 6 categories with 3 or fewer image representations, namely pizza box, polypropylene bag, battery, other plastic cup, carded blister pack, and plastified paper bag. By using the top 9 super categories, and merging all else into unlabeled litter, the dataset was modified to be more suitable for training and testing. They called this dataset TACO- 10.

Splitting:

The data was split into training, validation and testing. For this division we need to create new train, val and test JSON files from the existing one. We used an equation that takes the percentage we want to divide the images and then randomly select the amount we need of images to the 3 arrays. Finally, we divide the segmentation of each image to their category and create the JSON files.

The division was 80% training, 10% validation and 10% testing. Before that we kept 50 images on the side for inference.

Training images: 1200, Validation images: 125, Testing images: 125 ,and Inference images: 50

Training annotations: 3794, Validation annotations: 416, Testing annotations: 383, and Inference annotations: 191

Data splitting is random and is not too practical to apply to datasets of segmented annotations, since each data sample may contain multiple categories and images cannot be easily split because of the varying segmentation. This can result in having a split with no annotations of a certain category or very few. Paper [5] proposed Iterative stratification to solve this problem. It attempts to make categories present in every split. It works as follows; find the images where each category can be found, begin iteration with the category with the fewest image occurrences, distribute those images one by one to the set in most need of that sample (randomly in case of ties) into train, validation, test, and remove. Those images from all other categories. Re-sort the category lists by image quantity and continue this operation until all categories are processed.

Grouping:

- First grouping method:

The categories were combined into the below ones. One of the problems of this split is that there are categories with only 8 and 2 annotations. This way we would have some splits with no annotations of a certain type.

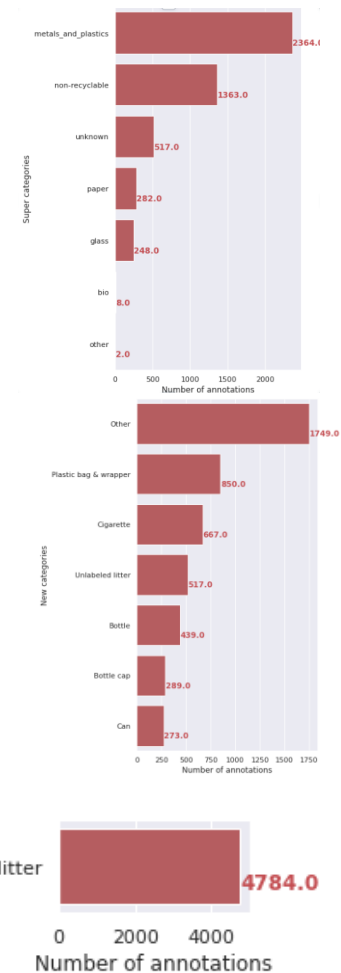
Many problems were faced while dividing them such as giving the new categories an ID and renumbering of the ones of the annotations.

- Second grouping method:

The categories first 6 categories only remained unchanged, and the rest were combined into 'other' group. Thus, now the group with the least number of annotations contains 273 annotations (Can category).

- Third grouping method:

The third type of grouping was 1 class grouping. All the images were in one category named 'litter'. We expect to get higher metric on the same model settings.



The experimental setup:

The environment used is COLAB. EDA was done there. First thing is to upload the data (images and JSON file) from our google drive and we unzip them. Then, we do the one of the grouping types we did before. This step outputs a new JSON. The next step is to split the data as discussed before. As a result, we get 3 output JSON files.

Detectron2 library is used. First, we import this library:

```
!pip install 'git+https://github.com/facebookresearch/detectron2.git' > /dev/null
```

Then, we register the data we have. There are several methods based on the format of our data. We used the below coco specific registering command as our data is in COCO format which makes it easy.

```
register_coco_instances('train',{},{},'/content/SplittedData/train.json', '/content/TACO/data')
register_coco_instances('test',{},{},'/content/SplittedData/test.json', '/content/TACO/data')
register_coco_instances('val',{},{},'/content/SplittedData/val.json', '/content/TACO/data')
```

The next step we use is dataset dictionary command which converts our data into a format used in the visualizing command.

```
dataset_dicts_train = get_detection_dataset_dicts(['train', 'test_val', 'test'])
```

Creating a metadata for our dataset is the next step which is done using the below command:

```
_dataset_metadata_train = MetadataCatalog.get('train')
_dataset_metadata_train.thing_colors = [cc['color'] for cc in builtin_meta.COCO_CATEGORIES]
```

The output of it is:

```
Metadata(evaluator_type='coco', image_root='/content/TACO/data', json_file='/content/SplittedData/train.json', name='train', thing_classes=['', '', '', '', '', '', ''], thing_colors=[[220, 20, 60], [119, 11, 32], [0, 0, 142], [0, 0, 230], [106, 0, 228], [0, 60, 100], [0, 80, 100], [0, 0, 70], [0, 0, 192], [250, 170, 30], [100, 170, 30], [220, 220, 0], [175, 116, 175], [250, 0, 30], [165, 42, 42], [255, 77, 255], [0, 226, 252], [182, 182, 255], [0, 82, 0], [120, 166, 157], [110, 76, 0], [174, 57, 255], [199, 100, 0], [72, 0, 118], [255, 179, 240], [0, 125, 92], [209, 0, 151], [188, 208, 182], [0, 220, 176], [255, 99, 164], [92, 0, 73], [133, 129, 255], [78, 180, 255], [0, 228, 0], [174, 255, 243], [45, 89, 255], [134, 134, 103], [145, 148, 174], [255, 208, 180], [197, 226, 255], [171, 134, 1], [109, 63, 54], [207, 138, 255], [151, 0, 95], [9, 80, 61], [84, 105, 51], [74, 65, 105], [166, 196, 102], [208, 195, 210], [255, 109, 65], [0, 143, 149], [179, 0, 194], [209, 99, 106], [5, 121, 0], [227, 255, 205], [147, 186, 208], [153, 69, 1], [3, 95, 161], [163, 255, 0], [119, 0, 170], [0, 182, 199], [0, 165, 120], [183, 130, 88], [95, 32, 0], [130, 114, 135], [110, 129, 133], [166, 74, 118], [219, 142, 185], [79, 210, 114], [178, 90, 62], [65, 70, 15], [127, 167, 115], [59, 105, 106], [142, 108, 45], [196, 172, 0], [95, 54, 80], [128, 76, 255], [201, 57, 1], [246, 0, 122], [191, 162, 208], [255, 255, 128], [147, 211, 203], [150, 100, 100], [168, 171, 172], [146, 112, 198], [210, 170, 100], [92, 136, 89], [218, 88, 184], [241, 129, 0], [217, 17, 255], [124, 74, 181], [70, 70, 70], [255, 228, 255], [154, 200, 0], [193, 0, 92], [76, 91, 113], [255, 180, 195], [106, 154, 176], [230, 150, 140], [60, 143, 255], [128, 64, 120], [92, 82, 55], [254, 212, 124], [73, 77, 174], [255, 160, 98], [255, 255, 255], [104, 84, 109], [169, 164, 131], [225, 199, 255], [137, 54, 74], [135, 158, 223], [7, 246, 231], [107, 255, 200], [58, 41, 149], [183, 121, 142], [255, 73, 97], [107, 142, 35], [190, 153, 153], [146, 139, 141], [70, 130, 180], [134, 199, 156], [209, 226, 140], [96, 36, 108], [96, 96, 96], [64, 170, 64], [152, 251, 152], [208, 229, 228], [206, 186, 171], [152, 161, 64], [116, 112, 0], [0, 114, 143], [102, 102, 156], [250, 141, 255]], thing_dataset_id_to_contiguous_id={0: 0, 1: 1, 2: 2, 3: 3, 4: 4, 5: 5, 6: 6})
```

Mainly it contains the dataset root directory, JSON file directory and other info like classes names, and some colors. (This file is needed later in deployment.)

We also need to set the num of classes we have using:

```
NUM_CLASSES = len(_dataset_metadata_train.thing_classes)
```

The most important step is the cfg or configuration file where all the training settings are set. We choose the registered train dataset and the registered test dataset. The below other thresholds are default.

```
cfg = get_cfg()

cfg.merge_from_file(model_zoo.get_config_file("COCO-InstanceSegmentation/mask_rcnn_R_101_FPN_3x.yaml"))

cfg.DATASETS.TRAIN = ("train",)
cfg.DATASETS.TEST = ("test_val",)
cfg.DATALOADER.NUM_WORKERS = 2
cfg.DATALOADER.SAMPLE_TRAIN = 'RepeatFactorTrainingSampler'
cfg.DATALOADER.REPEAT_THRESHOLD = 0.3
```

The second line in the above code imports the configuration file settings of the model we choose which is the Mask-RCNN R-101-FPN 3x. There are other configs as seen before, but this was our choice.


```

cfg.SOLVER.IMS_PER_BATCH = 2
cfg.SOLVER.BASE_LR = 0.0025
cfg.SOLVER.MAX_ITER = 3000
cfg.SOLVER.CHECKPOINT_PERIOD = 1000

# minimum image size for the train set
cfg.INPUT.MIN_SIZE_TRAIN = (800,)
# maximum image size for the train set
cfg.INPUT.MAX_SIZE_TRAIN = 1333
# minimum image size for the test set
cfg.INPUT.MIN_SIZE_TEST = 800
# maximum image size for the test set
cfg.INPUT.MAX_SIZE_TEST = 1333

```

Other important params are also set. Our image batch size is 2, base learning rate is 0.0025, max iterations or epochs is 3000 and checkpoint is after how many epochs a version of the model is saved. Moreover, we set the min image size as 800 and the max is 1333. This line uses `ResizeShortestEdge` function which scale the shorter edge to the given size, with a limit of `max_size` on the longer edge. If `max_size` is reached, then downscale so that the longer edge does not exceed `max_size`.

```

cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 512 #128
cfg.MODEL.ROI_HEADS.NUM_CLASSES = NUM_CLASSES

os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)

trainer = DefaultTrainer(cfg)
trainer.resume_or_load(resume=False)

```

The final parameter is the batch size per image of region of interest head. This is not the batch size we normally know. (It is a parameter that is used to sample a subset of proposals coming out of RPN to calculate cls and reg loss during training)

Note: After running the training we got an error indicating a repetition in some of the annotations id number. Annotations with id 309 and 4040 are repeated (only the id is repeated but they are for different images). They got converted to 4784 and 4785 respectively which solved the problem.

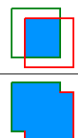
Metrics:

Intersection over Union (IoU)

IoU metric in object detection evaluates the degree of overlap between the ground(*gt*) truth and prediction(*pd*). The ground-truth and the prediction can be of any shape-rectangular box, circle, or even irregular shape). It is calculated as follows:

$$IoU = \frac{\text{area}(gt \cap pd)}{\text{area}(gt \cup pd)}$$

Diagrammatically, IoU is defined as follows (area of intersection divided by area of union between ground-truth and predicted box).

$$IoU = \frac{\text{area of overlap}}{\text{area of union}} = \frac{\text{area of intersection}}{\text{area of union}}$$


IoU ranges between 0 and 1 where 0 shows no overlap and 1 means perfect overlap between *gt* and *pd*. IoU is useful through thresholding, that is, we need a threshold (α , say) and using this threshold we can decide if a detection is correct or not.

Average Precision

$AP@α$ is Area Under the Precision-Recall Curve(AUC-PR) evaluated at $α$ IoU threshold. Formally, it is defined as follows

$$AP@α = \int_0^1 p(r) dr$$

$APα$ means that AP precision is evaluated at $α$ IoU threshold. If you see metrics like AP50 and AP75 then they just mean AP calculated at IoU=0.5 and IoU=0.75, respectively.

Mean Average Precision (mAP)

Remark (AP and the number of classes): AP is calculated individually for each class. This means that there are as many AP values as the number of classes (loosely). These AP values are averaged to obtain the metric: **mean Average Precision (mAP)**. Precisely, mean Average Precision (mAP) is the average of AP values over all classes.

$$mAP@α = \frac{1}{n} \sum_{i=1}^n AP_i \quad \text{for } n \text{ classes.}$$

(source: <https://towardsdatascience.com/on-object-detection-metrics-with-worked-example-216f173ed31e>)

```
Average Precision (AP):
AP          % AP at IoU=.50:.95 (primary challenge metric)
APIoU=.50  % AP at IoU=.50 (PASCAL VOC metric)
APIoU=.75  % AP at IoU=.75 (strict metric)
AP Across Scales:
APsmall    % AP for small objects: area < 322
APmedium   % AP for medium objects: 322 < area < 962
APlarge    % AP for large objects: area > 962
```

Results, discussion, and outcomes:

We will discuss in this section the models and results based on the 3 grouping methods:

Model 1 denotes using: Mask-RCNN_R_50_FPN_3x

Model 2 denotes using: Mask-RCNN_R_101_FPN_3x

		segm					
		AP	AP50	AP75	APs	APm	API
Group 1	Model 1	6.666	9.972	6.933	1.334	5.070	10.376
	Model 2	7.789	11.280	7.800	1.175	3.884	11.530
Group 2	Model 1	18.018	24.856	19.339	2.163	6.429	21.844
	Model 2	19.997	28.186	19.536	1.651	8.571	23.305
Group 3	Model 1	40.114	54.666	43.881	3.607	18.866	56.331
	Model 2	39.832	53.365	43.374	2.067	19.057	55.992
		bbox					

		AP	AP50	AP75	APs	APm	API
Group 1	Model 1	6.851	10.093	7.881	2.555	5.277	9.810
	Model 2	8.269	12.224	8.743	1.804	4.662	11.263
Group 2	Model 1	18.188	25.321	20.742	3.322	6.638	21.458
	Model 2	20.831	28.212	24.407	2.830	10.856	23.078
Group 3	Model 1	40.656	54.591	45.149	7.997	21.529	56.021
	Model 2	40.149	53.437	44.287	4.453	21.693	55.010

Note: For the augmentation we tried to use a custom trainer as follows:

```
from detectron2.data import transforms as T
train_augmentations = [
    T.RandomBrightness(0.5, 2),
    # T.RandomContrast(0.5, 2),
    T.RandomSaturation(0.5, 2),
    T.RandomFlip(prob=0.5, horizontal=True, vertical=False),
    # T.RandomFlip(prob=0.5, horizontal=False, vertical=True),
]

from detectron2.data import DatasetMapper, build_detection_train_loader
class MyTrainer(DefaultTrainer):
    # @classmethod
    # def build_evaluator(cls, cfg, dataset_name, output_folder=None):
    #     if output_folder is None:
    #         output_folder = os.path.join(cfg.OUTPUT_DIR, "inference")
    #     return COCOEvaluator(dataset_name, cfg, True, output_folder)
    @classmethod
    def build_train_loader(cls, cfg):
        # if "SemanticSegmentor" in cfg.MODEL.META_ARCHITECTURE:
        mapper = DatasetMapper(cfg, is_train=True, augmentations=train_augmentations)
        # else:
        #     mapper = None
        return build_detection_train_loader(cfg, mapper=mapper)

trainer = MyTrainer(cfg)
# trainer = DefaultTrainer(cfg)
trainer.resume_or_load(resume=False)
```

But after running it for training we were getting the size error on the colab:

```

9 frames
/usr/local/lib/python3.7/dist-packages/detectron2/layers/wrappers.py in forward(self, x)
    112
    113         x = F.conv2d(
--> 114             x, self.weight, self.bias, self.stride, self.padding, self.dilation, self.groups
    115         )
    116         if self.norm is not None:
RuntimeError: CUDA out of memory. Tried to allocate 976.00 MiB (GPU 0; 14.76 GiB total capacity; 10.65 GiB already allocated; 681.75 MiB free; 12.78 GiB reserved in total by PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid fragmentation.  See documentation for Memory Management and PYTORCH_CUDA_ALLOC_CONF
SEARCH STACK OVERFLOW
```

We tried a lot of suggested solutions and nothing helped. So we just tried the dataset as it is.

Deployment:

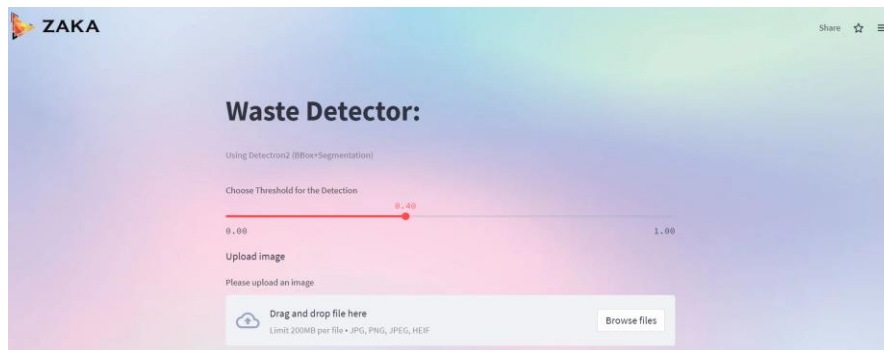
Deployment was done on Streamlit helps create web apps for data science and machine learning in a short time. It is compatible with major Python libraries such as scikit-learn, Keras, PyTorch, SymPy(latex), NumPy, pandas, Matplotlib etc.

Its commands are simple. For example, if we want to put a text we use `st.write('text')`. It is based on Flask and has many ready to use widgets. To deploy our model on their Cloud we need to upload the code to Github. 4 main files are needed. First is the requirements file which includes our dependencies.

10 lines (8 sloc) | 243 Bytes

```
1 pyyaml==5.1
2 torch==1.9.0
3 torchvision==0.10.0
4 streamlit
5
6 -f https://dl.fbaipublicfiles.com/detectron2/wheels/cpu/torch1.9/index.html
7 detectron2
```

Then, we need our main .py file where our code is written there. Third file is the model and the last one is the metadata of our data which was downloaded. Moreover, the model size is about 300 MB which can't be uploaded to github the traditional way. It was done using github LSF.



The user is free to choose the threshold and upload the image he wants.

A small test of the website:



Implications of the results on the industry/domain of the problem.

- This technology could save on costs or reduce expenses for disposal
- Increase the recycling process efficiency thus creating a better environment
- Autonomous mobile robot to detect, collect and recycle unwanted waste
- Automated waste recycling line

Potential directions for future work and improvements

- Perform "Iterative Stratification" data splitting based on annotations
- Add more photos (test the idea of combining datasets)
- Yolov7 segmentation
- Mobile app (Yolov7s or tiny - D2GO - EfficientDet-B0....)