Computer Vision and Deep Learning 2021

YOLOv4, YOLOv5, MaskRCNN for Recycling Codes Detection and Segmentation

Anna Boronina

Innopolis University

December 12, 2021

1 Introduction

Only 20 companies are responsible for 50% of waste production. To change that, people started to use supply-and-demand rule: if customers demand something, big companies will have to learn to supply it in order to stay on the market. Therefore, if people show that they care about the production of waste, big companies will either change their behavior or leave the market. One popular way to show passion for ecology is to recycle plastic, glass, aluminium, etc. Even if people split recyclable materials into several baskets, recycling companies do a second check because a few plastic cups mixed with paper may affect the new paper quality. So, to help big recycling companies process the amount of recycling materials they receive, I suggest to use state-of-the-art models to detect and even segment out the recycling codes: PAP (paper), POL (polyester), ALU (aluminium).
For the source code, go directly to 2.6

Class Name	Images Count	Objects Count
PAP	31	32
POL	36	37
ALU	29	29

Table 1: Original photos: classes statistics

Figure 1: Original images

Figure 2: Supervisely Annotations

2 Methodology

2.1 Data Acquisition, Augmentation, and Visualization

2.1.1 Data Acquisition and Annotation

I took the photos of the items I found at home: packs from crisps, cookies, cans from beans, etc. Table 1 shows the statistics of number of images per class.
I used Supervisely to annotate the data. Fig. 1 presents the original photos and Fig. 2 presents these photos after annotation process.

2.1.2 Problems with data

1. For every item, I took several photos from different angle. The intention was to make the models view-angle-invariant. The side effect of it is that after augmentation (which included increase in size) repeated images were repeated even more times even though they went through cutout, color change, brightness change, etc.

2. Another problem is that I made sure there was perfect lightning when taking photos. The intention was to make sure all recycling codes are very easily readable. It was before I knew YOLOs do cutout augmentation. Nevertheless, I hoped that augmentations (blurring, cutout, noise) will fix my mistake of assuming the model needs perfect images.

A conclusion about whether it affected the training or not will be made at the end.

2.1.3 Data Augmentation for YOLOs

I chose Roboflow to transform supervisely dataset from supervisely format to the data formats YOLOs need. I used Darknet format for YOLOv4 and yolov5-pytorch format for YOLOv5. Roboflow lets its users preprocess, augments and train-test

Figure 3: Roboflow-augmentated images

split the data before exporting it in the necessary formats. Changes in the dataset are the following:
Preprocessing

- Auto-orient

Augmentations

- Outputs per training example: 3
- Rotation: Between -45° and +45°
- Shear: ±20° Horizontal, ±20° Vertical
- Hue: Between -180° and +180°
- Saturation: Between -50% and +50%
- Brightness: Between -30% and +30%
- Blur: Up to 2.75px

Even though Supervisely annotations contain polygons, Roboflow turns them into bounding boxes, which is what YOLOs need. Fig. 3 presents the examples of Roboflow-augmented images. In fact, they are a bit more darkened than they really are because of the highlightened bounding boxes.

2.2 Data Augmentation for MaskRCNN

This time I used Supervisely for augmentations and size increase because MaskR-CNN requires different format: bounding boxes, segmentation polygons, etc. I added augmentations by using Supervisely's DTL language to write a configuration and run the task. The result of this is a new dataset of 234 images. I tried to make these augmentations as close as possible to those I made for YOLOs:
Augmentations:

- Resize: 700x700
- Rotate: Between -180° and +180°
- Gaussian Blur: sigma between 0.5 and 2
- Contrast: between 0.5 and 2

Class Name	Images Count	Objects Count
PAP	93	96
POL	108	111
ALU	87	87

Table 2: MaskRCNN dataset: classes statistics

Figure 4: Roboflow-augmented images

- Brightness: between -50 and 50
- Random Color

Fig. 4 shows examples from the dataset and Table 2 shows classes statistics of the dataset for MaskRCNN - basically, all numbers got multiplied by 3 comparing to Table 1.

2.3 YOLOv4-tiny

I used the dataset exported in Darknet format from Roboflow. Some statistics regarding training:

- Configuration: batch=64; subdivisions=20; image=(416, 416); lr=0.00261; momentum=0.9; decay=0.0005;
- YOLOv4 trained for 6000 iterations
- Last accuracy: 80%
- Best accuracy: 84%
- The model saw 360.000 images
- Total detection time for test set: 1s
- Average IoU for confidence threshold 25%: 57%

Class Name	Average Precision	True Positive (TP)	False Positive (FP)
ALU	70.24	4	0
PAP	67.26	5	3
POL	100	8	0

Table 3: YOLOv4: statistics

Figure 5: YOLOv4 and YOLOv5 input

Figure 6: YOLOv4 output

Table 3 shows the test statistics of YOLOv4. As I wrote before, POL class had the highest number of images and PP recycling code is always associated with number 5, whereas PAP and ALU codes could be associated with multiple numbers. Probably that is why AP for POL class is 100%.
Fig. 5 shows the type input images YOLOv4 received, and Fig. 6 shows the output of YOLOv4 with its best weights. YOLOv4 made correct predictions except the last image in which it was not sure whether it is a POL or ALU class, while in fact it is POL.

2.4 YOLOv5-tiny

I used tiny-config for training YOLOv5. Unlike YOLOv4, I trained it with and without initial weights. YOLOv5 had the same images for input as YOLOv4, so see Table 5.
Table 4 shows the comparison between YOLOv5 and pretrained YOLOv5. Both YOLOs were trained on my dataset. YOLOv5 with pretrained weights has higher precision and mAP but a bit smaller recall. These are not very good metrics, as they are the last ones. Due to early stopping I did not receive a proper report. Because of that, I attach the graphs of metrics changes: refer to Table **??**. I do not see much difference in training progress and all metrics stopped increasing after epoch 800.
Figs. 8 and 9 show the output of YOLOv5 without and with initial weights respectively. For both evaluation I used best weights. I believe that pretrained YOLOv5

Metric	YOLOv5	Pretrained YOLOv5
Epochs	1400	1750
Last precision	86%	92%
Last recall	47%	45%
Last mAP	29%	48%
Time spent	1h40m	2h16m

Table 4: YOLOv5 vs pretrained YOLOv5

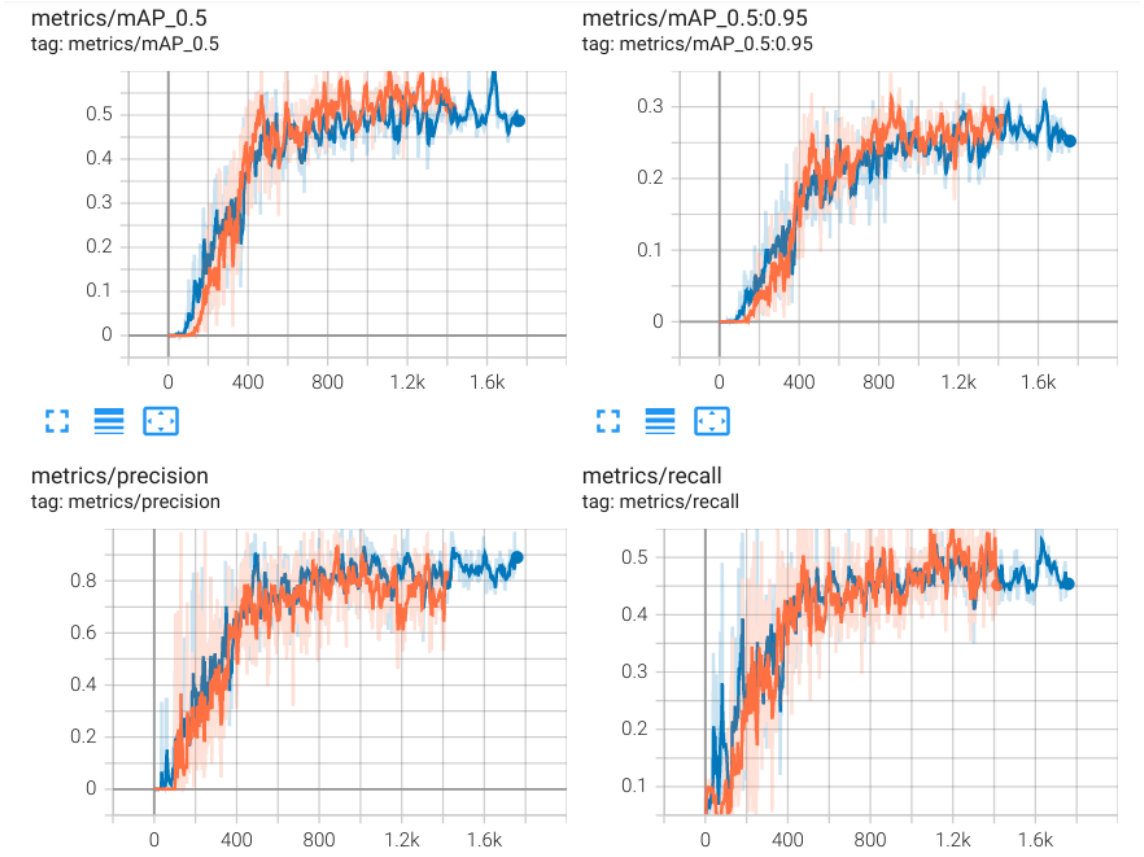

Figure 7: YOLOv5: orange; Pretrained YOLOv5: blue

Figure 8: YOLOv5 output

Figure 9: Pretrained YOLOv5 output

AP	AP50	AP75
90%	100%	100%

Table 5: MaskRCNN: Average Precision at different thresholds

Class Name	AP
PAP	90.6%
POL	88.7%
ALU	90%

Table 6: MaskRCNN: Average Precision for 3 classes

became more accurate in its predictions, because there are less false positives.

2.5 MaskRCNN

As I mentioned before, MaskRCNN requires a different type of data: both bounding boxes and segmentation polygons. To do that, I wrote a *.py* script that takes a path to the dataset in supervisely format and creates two *.json* files: *train.json* and *valid.json*. These two files are fed into the model.
To train the model, I used Detectron2 environment for configuration.

- I used pretrained weights from COCO
- Learning rate: 0.0002
- Number of iterations: 4000
- Decrease learning rate by 0.5 at iterations: 2800, 3600
- Images size is 704x704 (must be a multiple of 32)
- Finished after 58 minutes - very fast!

2.5.1 Metrics

Figs. 5 and 6 show statistics of average precision for different confidence thresholds and classes respectively.
Since MaskRCNN is an extension of FasterRCNN, Detectron2 offers metrics for both MaskRCNN and FastRCNN. MaskRCNN predicts masks, while FastRCNN predicts bounding boxes Figs. 10 and 11 show statistics throughout the training process for FastRCNN and MaskRCNN respectively.

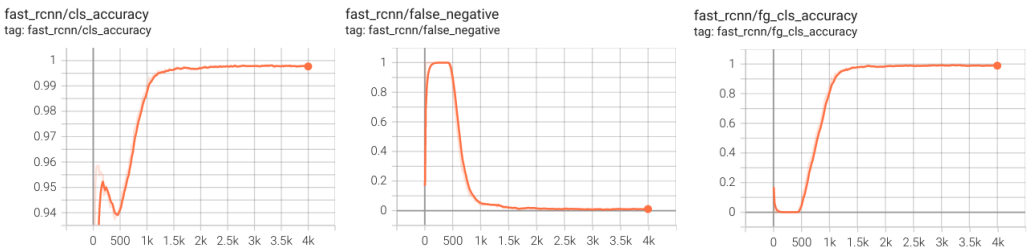

Figure 10: FastRCNN statistics

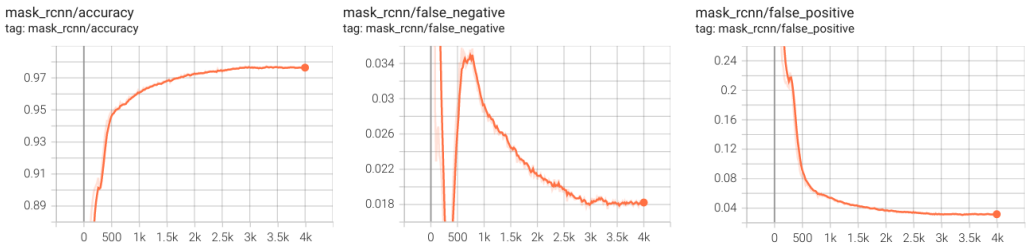

Figure 11: MaskRCNN statistics

Figure 12: MaskRCNN predictions

2.5.2 Outputs

Fig. 12 shows predictions of MaskRCNN. All of them have confidence above 95% and all of them are predicted correctly without any false positives.

2.6 Source Code

GitHub repository. README contains all the necessary instructions on how to view the code and the results in the most effective way. I highly advise to view Jupyter Notebooks in Google Colab by following the links at the top of README.

3 Results

Table **??** indicates that MaskRCNN showed the best results across 3 out of 4 metrics.

Metric	YOLOv4	YOLOv5	MaskRCNN
AP or P	85%	**92%**	89.7%
Recall	77%	45%	**95%**
mAP	0.79%	48%	**96%**
Accuracy	80%	-	**100%**
Training time	1h15m	1h40min	58min

Table 7: Comparison of YOLOv4, YOLOv5, and MaskRCNN

Conclusions

I present a list of conclusions and opinions about these models.

- MaskRCNN was the most comfortable to work with: its training process output was informative and concise.
- MaskRCNN trains with Detectron2 which is the last and actively maintained framework.
- MaskRCNN was the fastest to train - took only 58 minutes to finish 4000 iterations, whereas YOLOv5 took twice as much for 2000 iterations.
- MaskRCNN showed the best results on the dataset. If I wanted to deploy such solution in production, I would definitely go with MaskRCNN.
- MaskRCNN and YOLOs trained on slightly different datasets, so maybe it could explain such difference in performance.
- Configuration of MaskRCNN is the most understandable. YOLOs' tiny configs let me change strides and filters, whereas Detectron2 just lets me extend or overwrite existing configuration.
- It is easy to connect a backbone to MaskRCNN in case there is a need for it.
- Even though experiments and research papers claim that YOLOv4 and YOLOv5 are state-of-the-art networks, they did not show good results on my dataset.
- At the beginning of the report I mentioned that I tried to make my dataset perfect: good light while taking photos, no blur. Of course it is wrong because I want my model to perform regardless of lightning. Yet I think that augmentations I applied before and during training helped to fix the problem.

Acknowledgements

I would like to acknowledge the constribution of our lecturer, Imad Eddine Ibrahim BEKKOUCH who supported me during the whole process and gave advice for training MaskRCNN.