Replication Study with Google Colab

HuanNing, New Jersey Institute of Technology

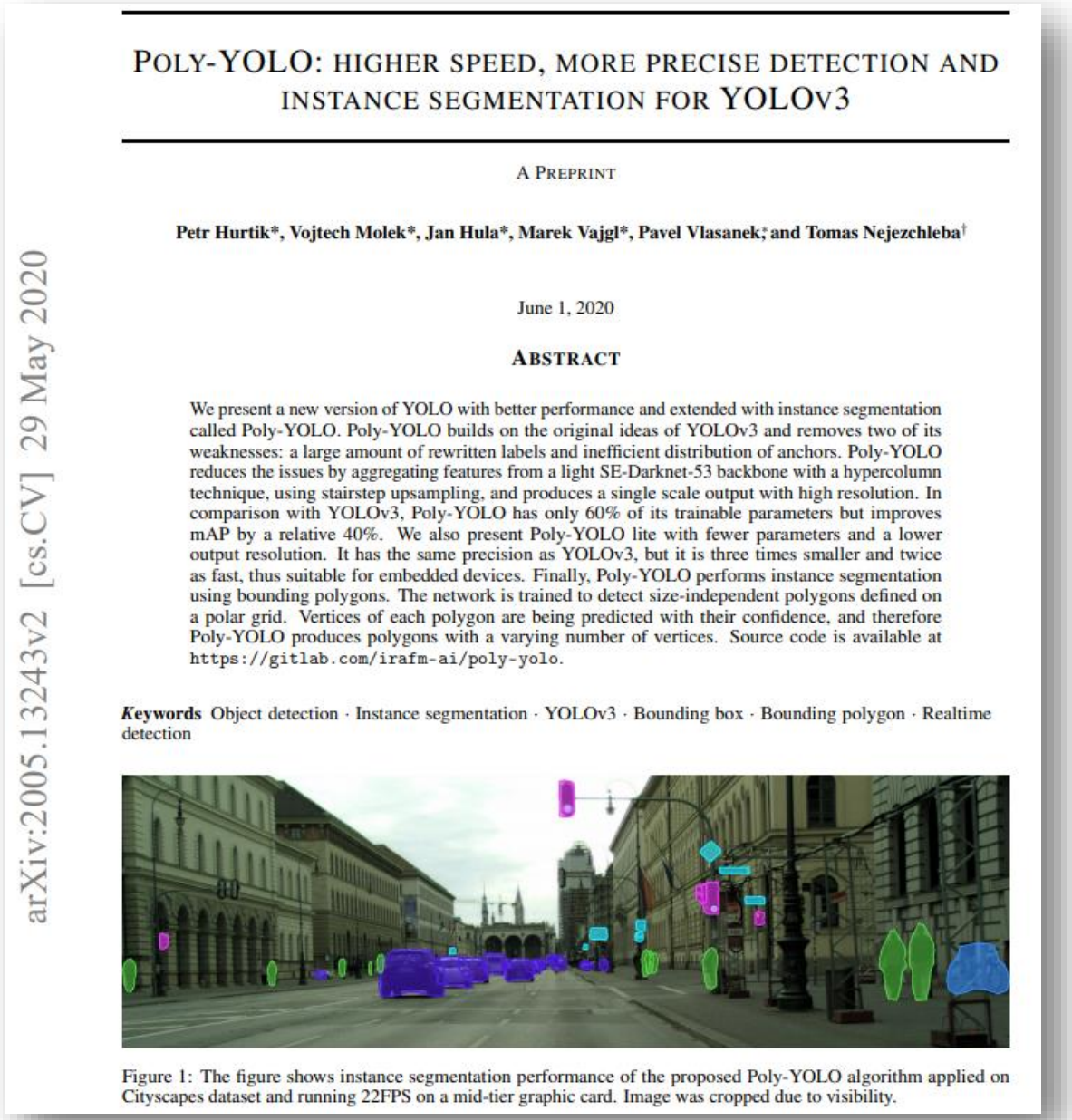
XinyueYe, Urban Data Science Lab, TAMU

June 26, 2020

A case study based on Google Colab and GitLab

Topics:

1. Paper introduction
2. Potential Extension
3. Experiment



POLY-YOLO: HIGHER SPEED, MORE PRECISE DETECTION AND INSTANCE SEGMENTATION FOR YOLOV3

A PREPRINT

Petr Hurtik*, Vojtech Molek*, Jan Hula*, Marek Vajgl*, Pavel Vlasanek‡ and Tomas Nejezchleba†

June 1, 2020

arXiv:2005.13243v2 [cs.CV] 29 May 2020

ABSTRACT

We present a new version of YOLO with better performance and extended with instance segmentation called Poly-YOLO. Poly-YOLO builds on the original ideas of YOLOv3 and removes two of its weaknesses: a large amount of rewritten labels and inefficient distribution of anchors. Poly-YOLO reduces the issues by aggregating features from a light SE-Darknet-53 backbone with a hypercolumn technique, using stairstep upsampling, and produces a single scale output with high resolution. In comparison with YOLOv3, Poly-YOLO has only 60% of its trainable parameters but improves mAP by a relative 40%. We also present Poly-YOLO lite with fewer parameters and a lower output resolution. It has the same precision as YOLOv3, but it is three times smaller and twice as fast, thus suitable for embedded devices. Finally, Poly-YOLO performs instance segmentation using bounding polygons. The network is trained to detect size-independent polygons defined on a polar grid. Vertices of each polygon are being predicted with their confidence, and therefore Poly-YOLO produces polygons with a varying number of vertices. Source code is available at `https://gitlab.com/irafm-ai/poly-yolo`.

Keywords Object detection · Instance segmentation · YOLOv3 · Bounding box · Bounding polygon · Realtime detection

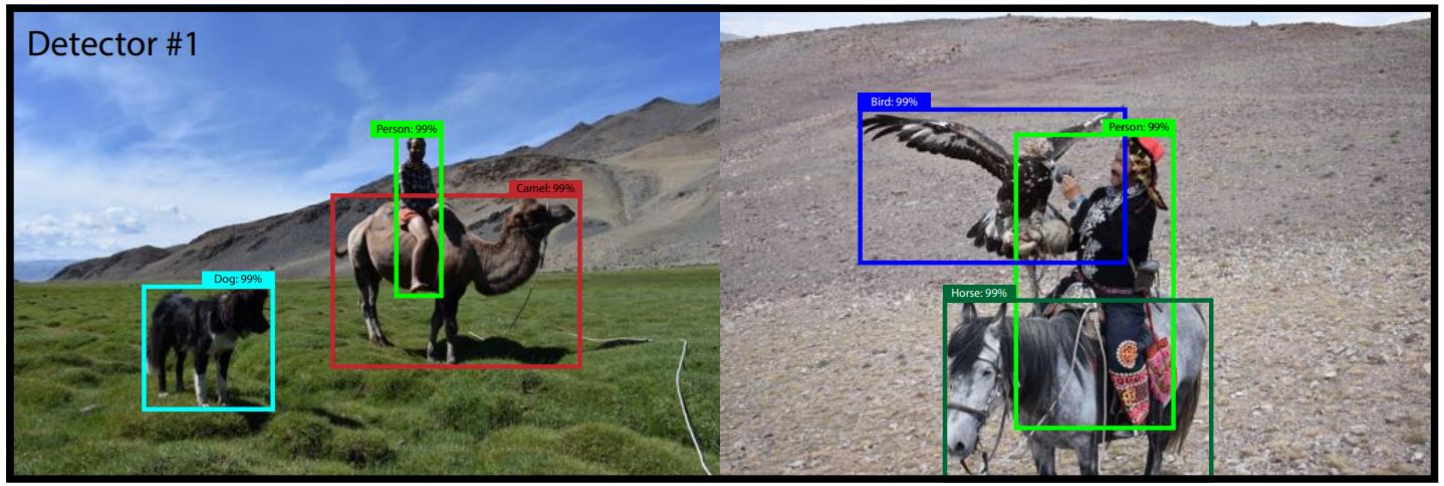
Figure 1: The figure shows instance segmentation performance of the proposed Poly-YOLO algorithm applied on Cityscapes dataset and running 22FPS on a mid-tier graphic card. Image was cropped due to visibility.

Replicated paper:

Poly-YOLO: higher speed, more precise detection and instance segmentation for YOLOv3

Hurtik P, Molek V, Hula J, et al. Poly-YOLO: higher speed, more precise detection and instance segmentation for YOLOv3[J]. arXiv preprint arXiv:2005.13243, 2020.

YOLO

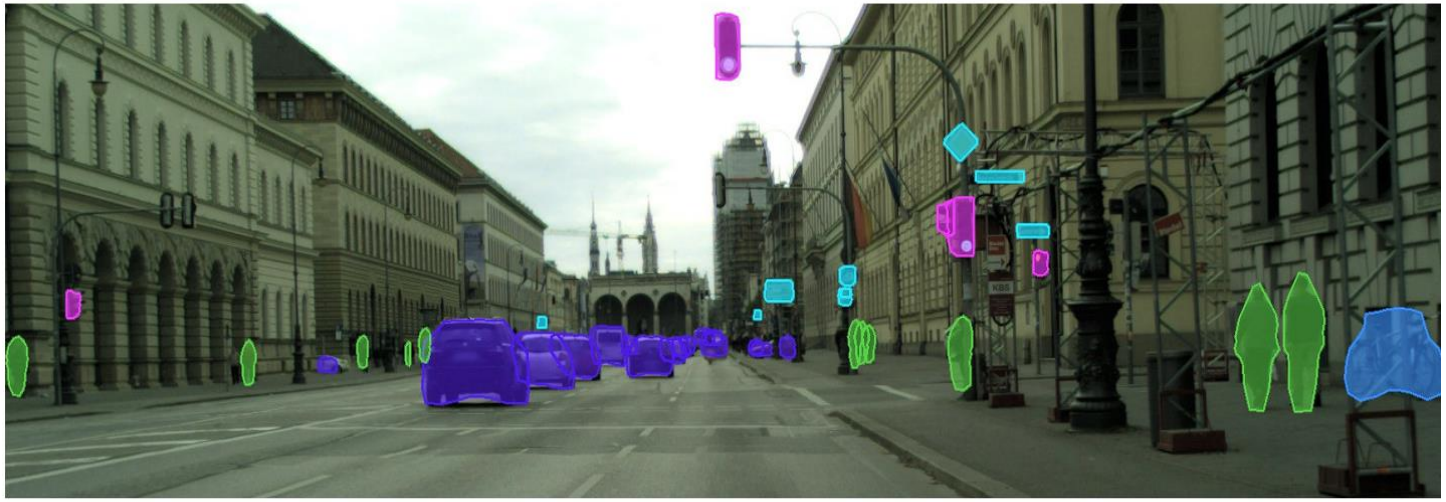


Redmon J, Divvala S, Girshick R, et al. You only look once: Unified, real-time object detection[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 779-788.

Redmon J, Farhadi A. YOLO9000: better, faster, stronger[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2017: 7263-7271.

Redmon J, Farhadi A. Yolov3: An incremental improvement[J]. arXiv preprint arXiv:1804.02767, 2018.

Poly-YOLO: an improved YOLOv3



Object Detector:

Input: an image

Output: object bounding box and categories

Hurtik P, Molek V, Hula J, et al. Poly-YOLO: higher speed, more precise detection and instance segmentation for YOLOv3[J]. arXiv preprint arXiv:2005.13243, 2020.

Poly-YOLO: higher speed, more precise detection and instance segmentation for YOLOv3

Few parameters

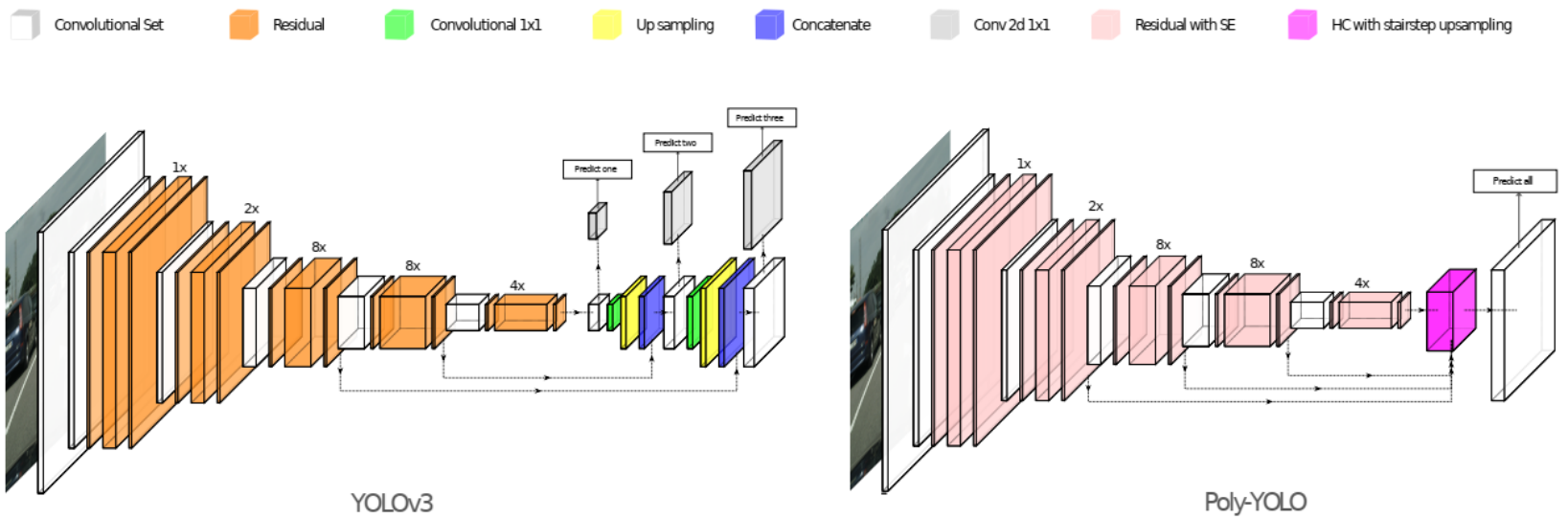


Figure 6: A comparison of YOLOv3 and Poly-YOLO architecture. Poly-YOLO uses less convolutional filters per layer in the feature extractor part and extends it by squeeze-and-excitation blocks. The heavy neck is replaced by a lightweight block with hypercolumn that utilizes a stairstep for upsampling. The head now uses single instead of three outputs and has a higher resolution. In summary, Poly-YOLO has 40% less parameters than YOLOv3 while producing more precise predictions.

Poly-YOLO: higher speed, more precise detection and instance segmentation for YOLOv3

More Grids
Staristep upsampling
Hypercolumn

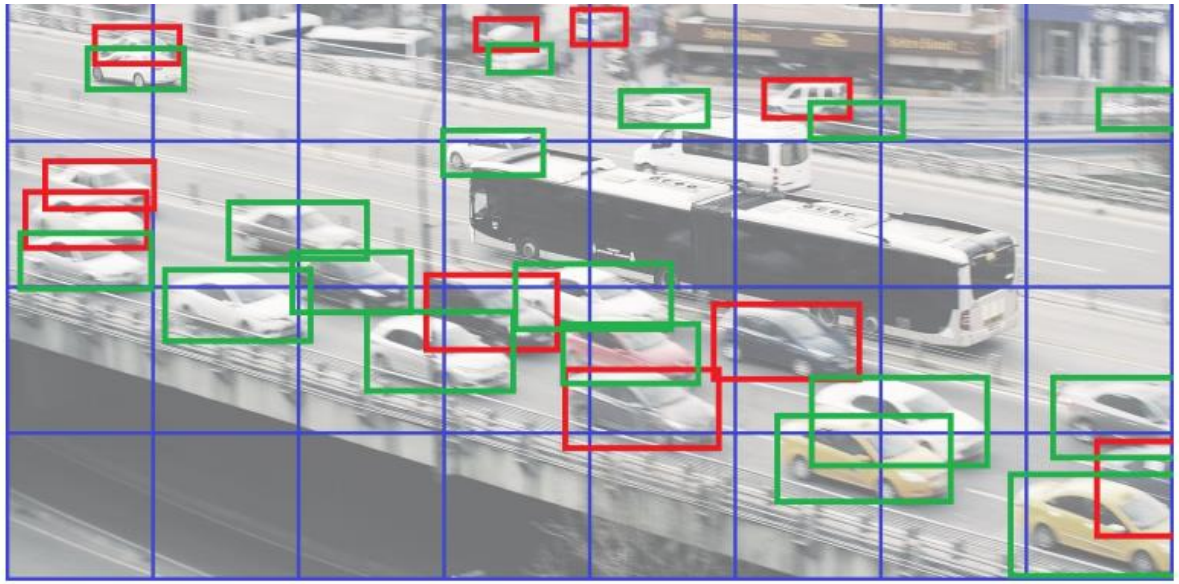


Figure 5: The image illustrates the label rewriting problem for the detection of cars. A label is rewritten by other if centers of two boxes (with the same anchor box) belong to the same cell. In this illustrative example, blue denotes grid, red rewritten label, and green preserved label. Here, 10 labels out of 27 are rewritten, and the detector is not trained to detect them.

Poly-YOLO: higher speed, more precise detection and instance segmentation for YOLOv3

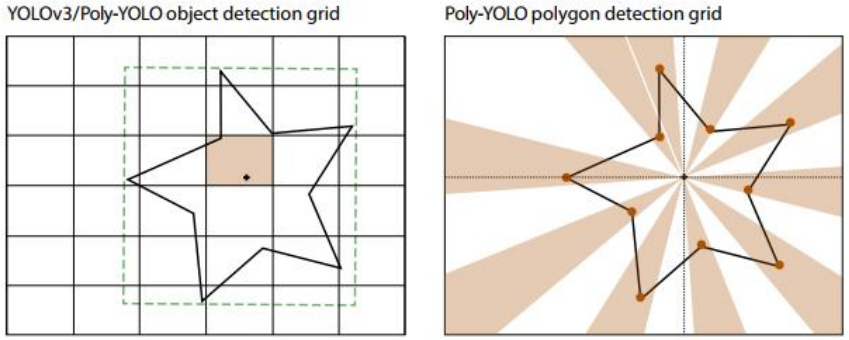


Figure 9: The image illustrates grids used in Poly-YOLO. Left: the rectangular grid, which is taken from YOLOv3. A cell where an object's bounding box has its center predicts its bounding box coordinates. Right: the grid based on circular sectors used in Poly-YOLO for the detection of vertices of the polygon. The center of the grid coincides with the center of the object's bounding box. Each circular sector is then responsible for detecting polar coordinates of the particular vertex. Sectors, where no vertex is present, should yield confidence equal to zero.

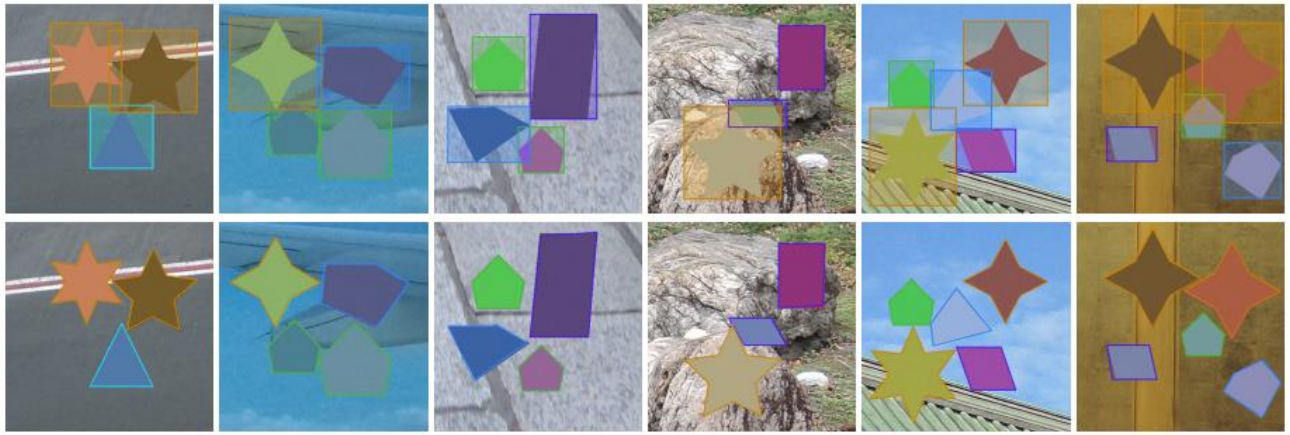


Figure 10: Comparison of Poly-YOLO bounding box detection (top) and Poly-YOLO bounding polygon detection (bottom).

Polygons of a varying number of vertices

Results

Table 2: The results of the involved algorithms for bounding box detection and instance segmentation on the three datasets.

Method	Backbone	Resolution	#Parameters	Instance segment.	Box AP	Box AP_{50}	Box AP_{75}	Mask AP	Mask AP_{50}	Mask AP_{75}	FPS
PERFORMANCE ON THE SIMULATOR DATASET											
RetinaNet	ResNet-50 FPN	608×800	36,276,717	✗	0.475	0.714	0.487	–	–	–	25.0*
YOLOv3	Darknet-53	608×800	61,576,342	✗	0.305	0.699	0.220	–	–	–	21.2
Poly-YOLO	SE-Darknet-53	608×800	37,196,958	✗	0.413	0.735	0.408	–	–	–	22.0
Poly-YOLO	SE-Darknet-53 lite	416×576	16,545,766	✗	0.322	0.661	0.258	–	–	–	58.6
Mask R-CNN	ResNet-50	448×448	44,662,942	✔	0.389	0.664	0.414	0.203	0.452	0.157	15.8
Poly-YOLO	SE-Darknet-53	608×800	37,446,438	✔	0.435	0.745	0.445	0.345	0.731	0.272	19.6
Poly-YOLO	SE-Darknet-53 lite	416×576	16,712,302	✔	0.377	0.694	0.348	0.298	0.675	0.270	52.7
PERFORMANCE ON THE CITYSCAPES DATASET											
RetinaNet	Resnet-50 FPN	608×1216	36,504,912	✗	0.224	0.379	0.231	–	–	–	21.0*
YOLOv3	Darknet-53	416×832	61,640,962	✗	0.106	0.266	0.061	–	–	–	26.3
Poly-YOLO	SE-Darknet-53	416×832	37,238,538	✗	0.168	0.344	0.141	–	–	–	26.5
Poly-YOLO	SE-Darknet-53 lite	320×608	16,573,522	✗	0.104	0.231	0.080	–	–	–	46.8
Mask R-CNN	Resnet-50	1024×1024	44,722,144	✔	0.164	0.318	0.151	0.069	0.202	0.031	6.2
Poly-YOLO	SE-Darknet-53	416×832	37,488,018	✔	0.129	0.273	0.105	0.087	0.240	0.046	21.9
Poly-YOLO	SE-Darknet-53 lite	320×608	16,740,058	✔	0.114	0.253	0.091	0.078	0.217	0.044	37.2
PERFORMANCE ON THE INDIA DRIVING DATASET											
RetinaNet	Resnet-50 FPN	608×1080	36,546,402	✗	0.221	0.357	0.230	–	–	–	19.8*
YOLOv3	Darknet-53	448×800	61,646,347	✗	0.117	0.267	0.089	–	–	–	23.9
Poly-YOLO	SE-Darknet-53	448×800	37,242,003	✗	0.152	0.304	0.137	–	–	–	25.5
Poly-YOLO	SE-Darknet-53 lite	352×608	16,575,835	✗	0.125	0.260	0.105	–	–	–	46.7
Mask R-CNN	Renset-50	1024×1024	44,732,908	✔	0.175	0.300	0.177	0.098	0.217	0.077	7.5
Poly-YOLO	SE-Darknet-53	448×800	37,491,483	✔	0.145	0.288	0.134	0.115	0.267	0.083	20.6
Poly-YOLO	SE-Darknet-53 lite	352×608	16,742,371	✔	0.131	0.263	0.119	0.101	0.239	0.074	37.1

Poly-YOLO: higher speed, more precise detection and instance segmentation for YOLOv3

Pros

1. Fast, precise object detection.
2. Real-time instance segmentation.
3. Return contours, good for some circumstances.

Cons

1. Inaccuracy contours.
2. Allowing one vertex in a polar cell.
3. Lack results on other datasets such as COCO.

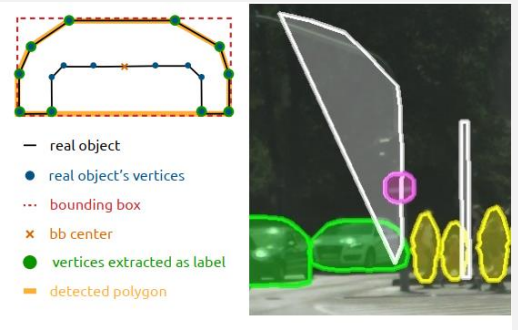


Figure 15: Left:A scheme of label creation for a problematic object, where the limitation appear. Right: impact on a
real predictions, see the lamp object.

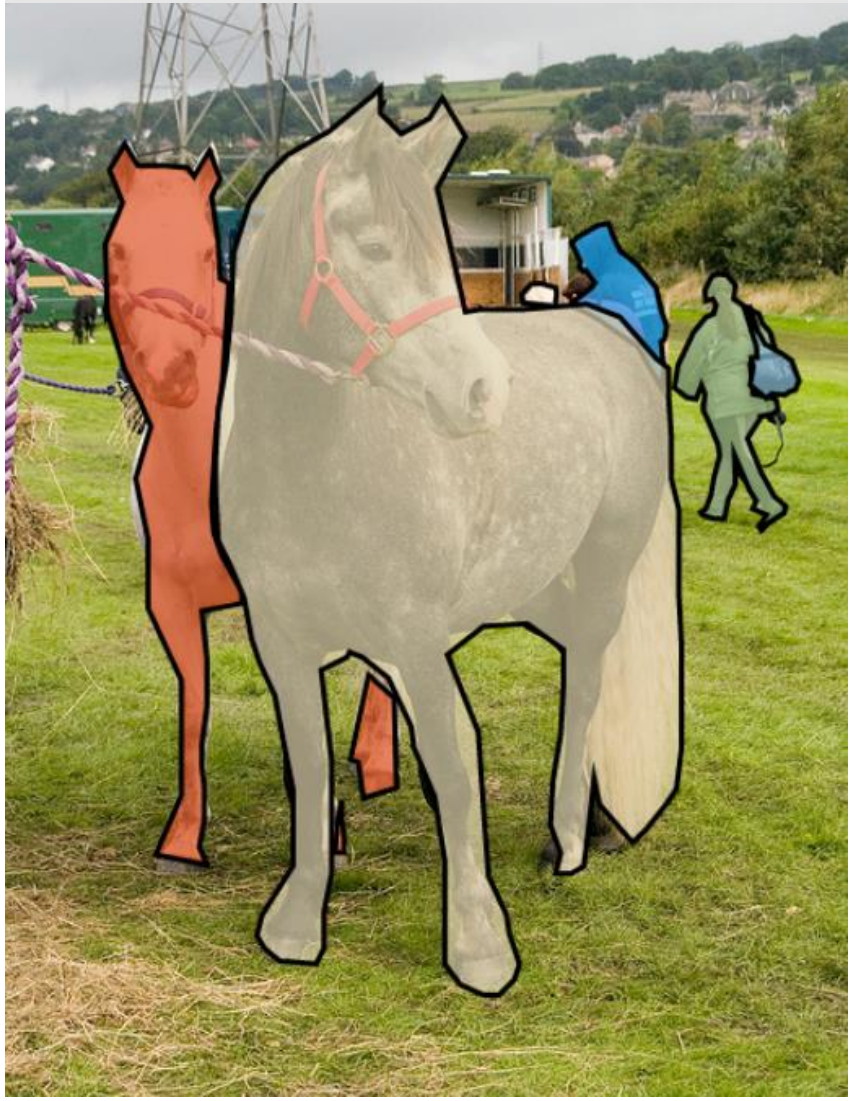
Topics:

A simple example based on Google Colab and GitLab.

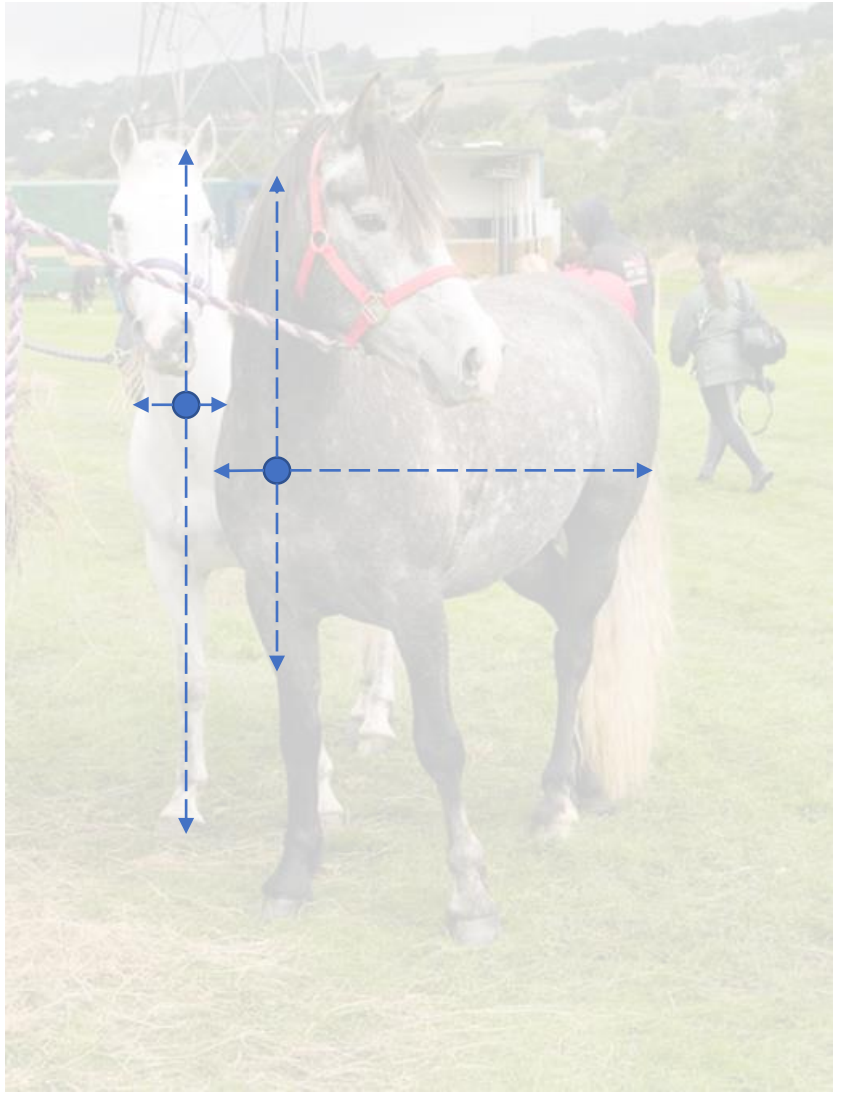
1. Paper introduction
2. Potential Extension
3. Experiment

Potential extension: field-based mask detection

Masks of two horses



boundary fields of two pixels



1) Calculate 4-direction boundary fields for pixels.

Direction: towards nearest point on boundary.

Value: distance.

Pixel

Non-nearest direction

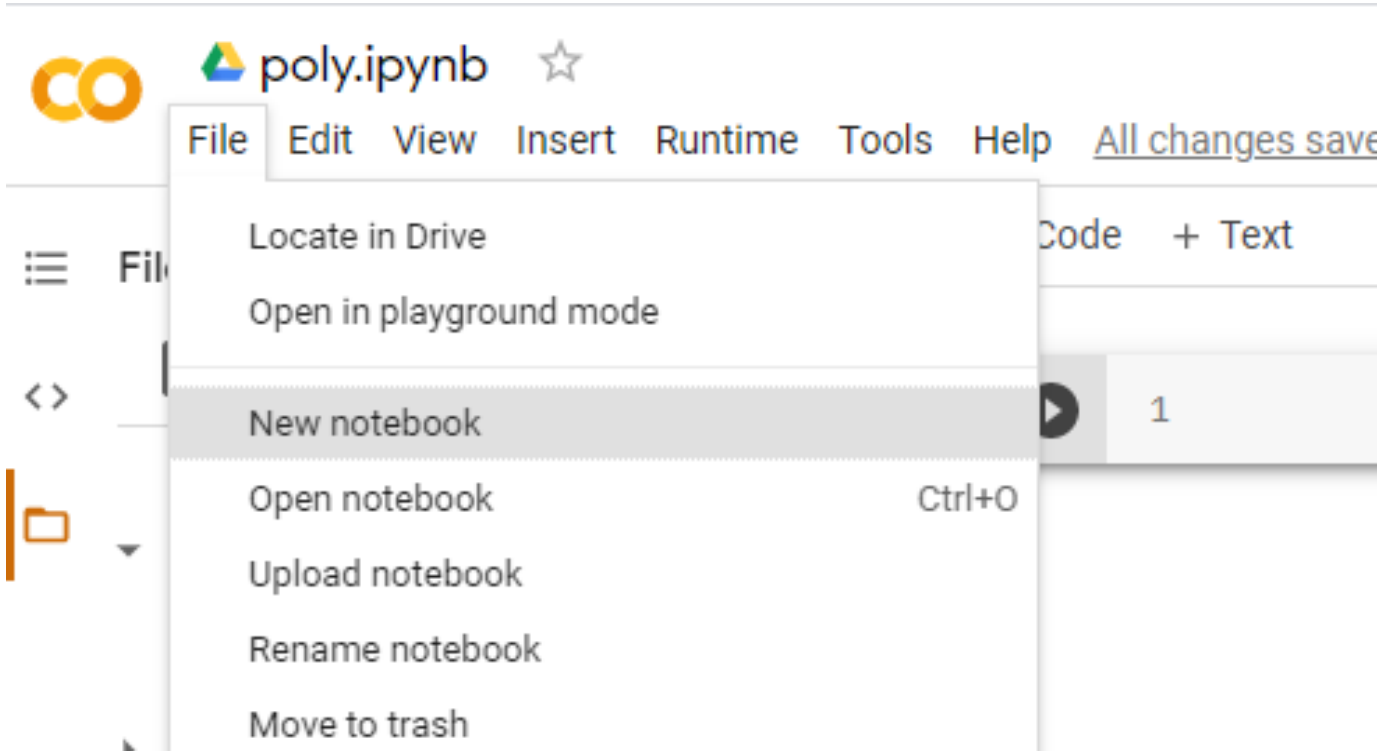
Nearest direction

2) Restore masks based on boundary field.

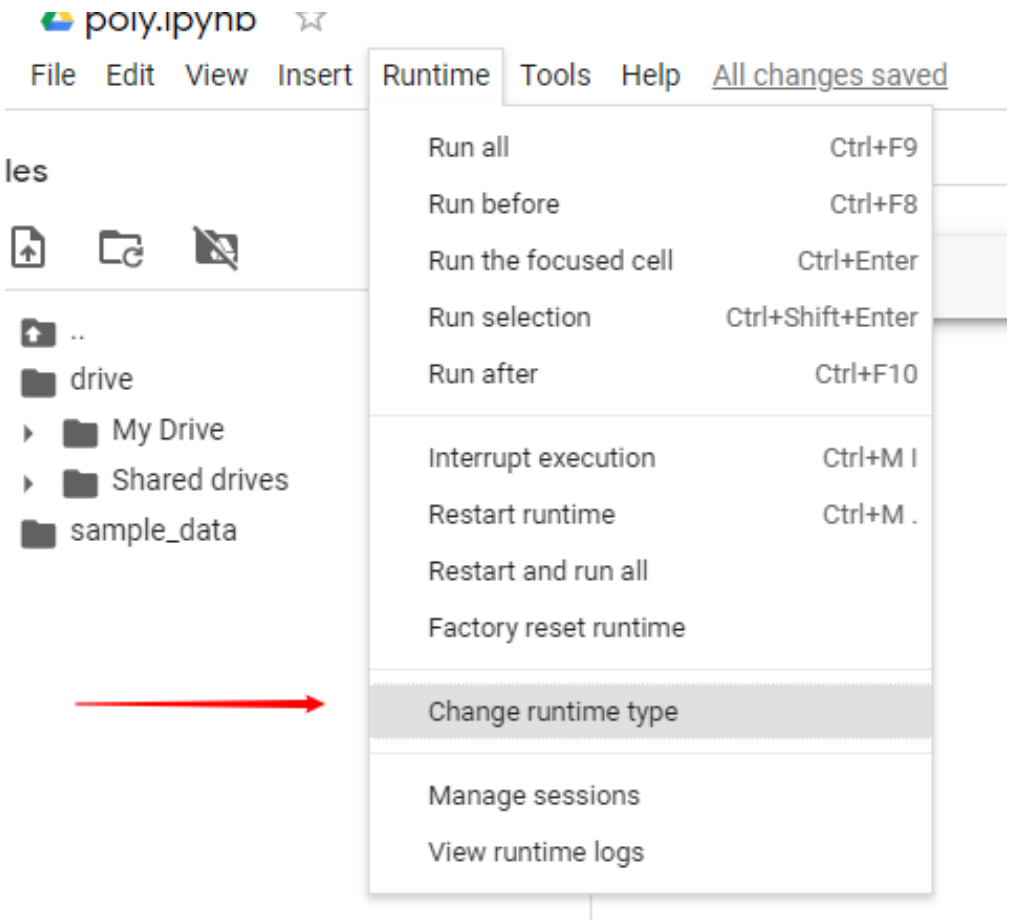
Overall workflows

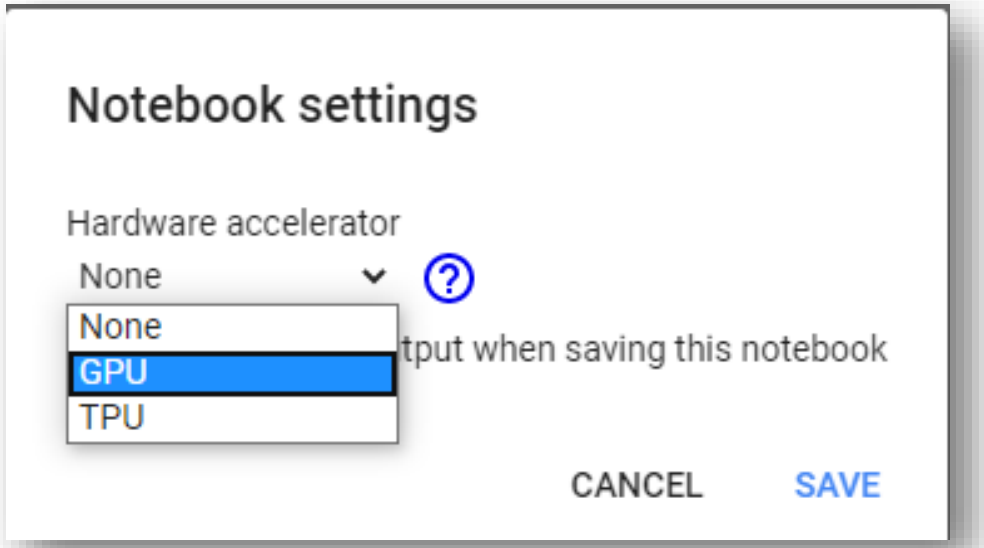
1. Create a new notebook in Colab.
2. Clone repository from GitLab.
3. Config paths of datasets.
4. Debug.

1. Create a new notebook in Colab (https://colab.research.google.com/).



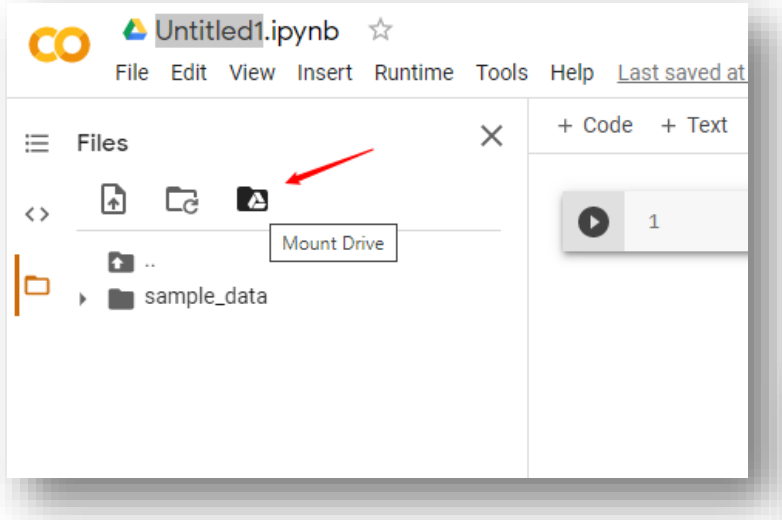
Change runtime (if want to use GPU/TPU).

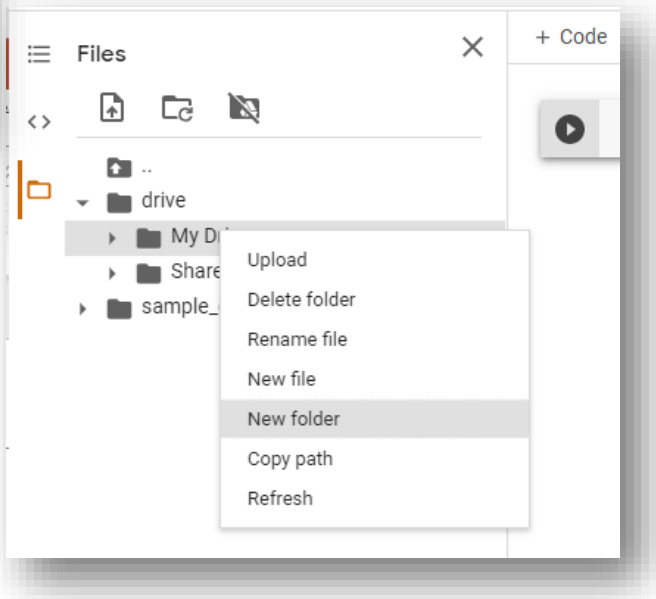




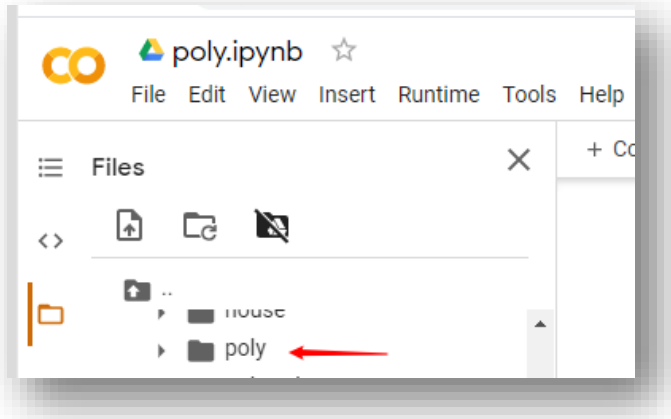
2. Clone a repository from GitLab.

Connect to your Google Drive for permanent data storage.



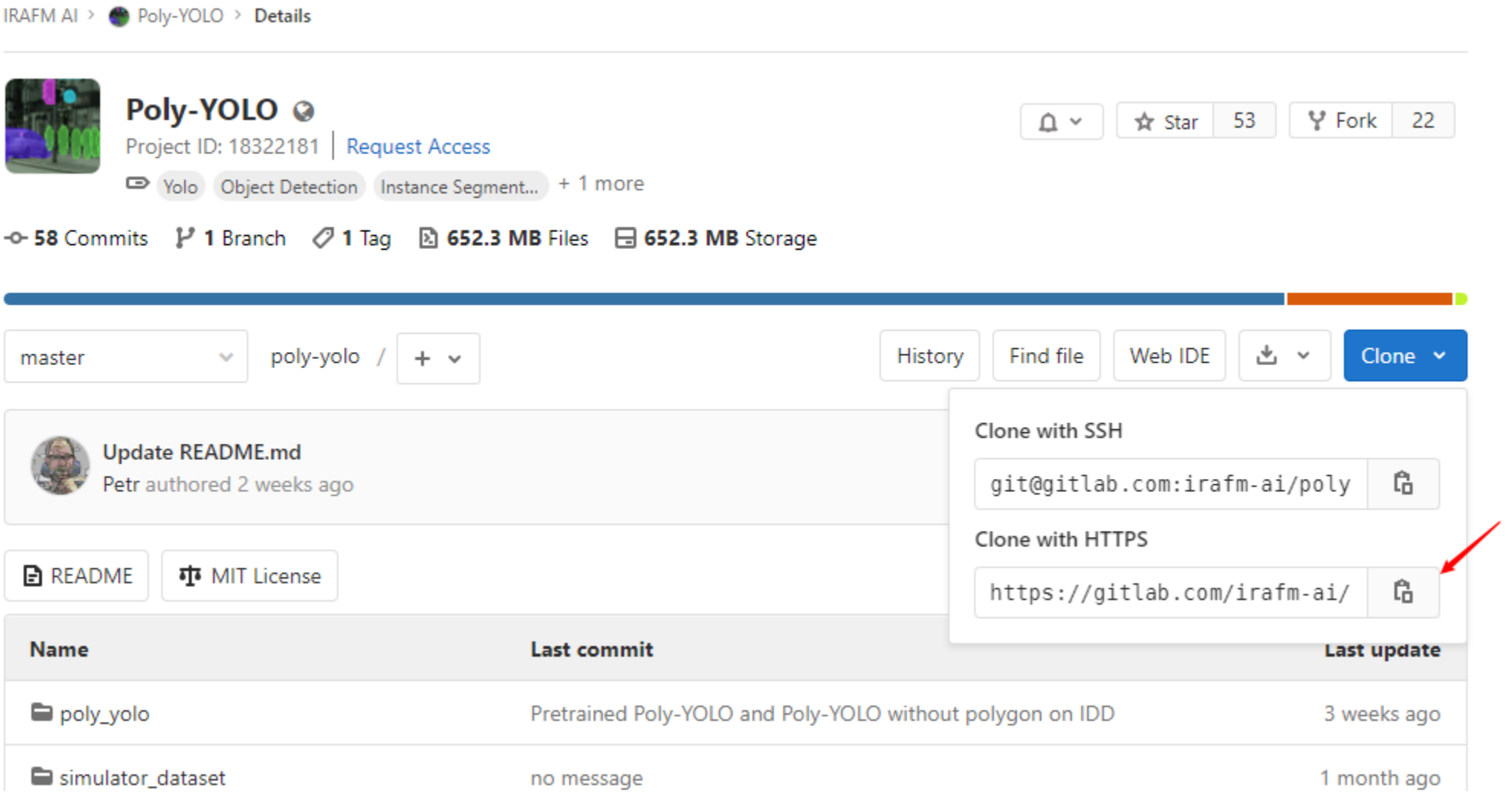


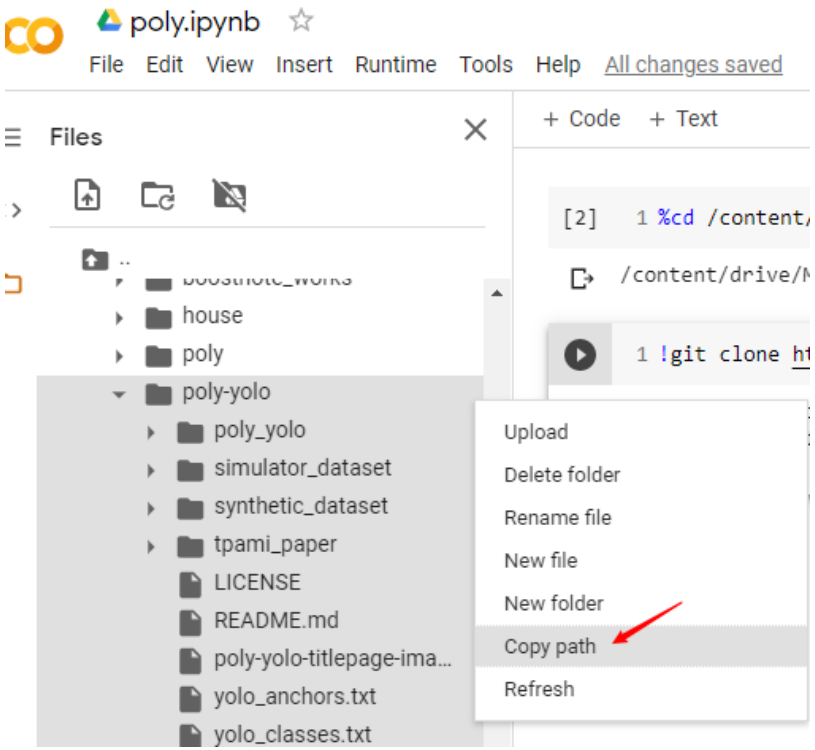
Create a new directory.



Copy the Git URL.

https://gitlab.com/irafm-ai/poly-yolo





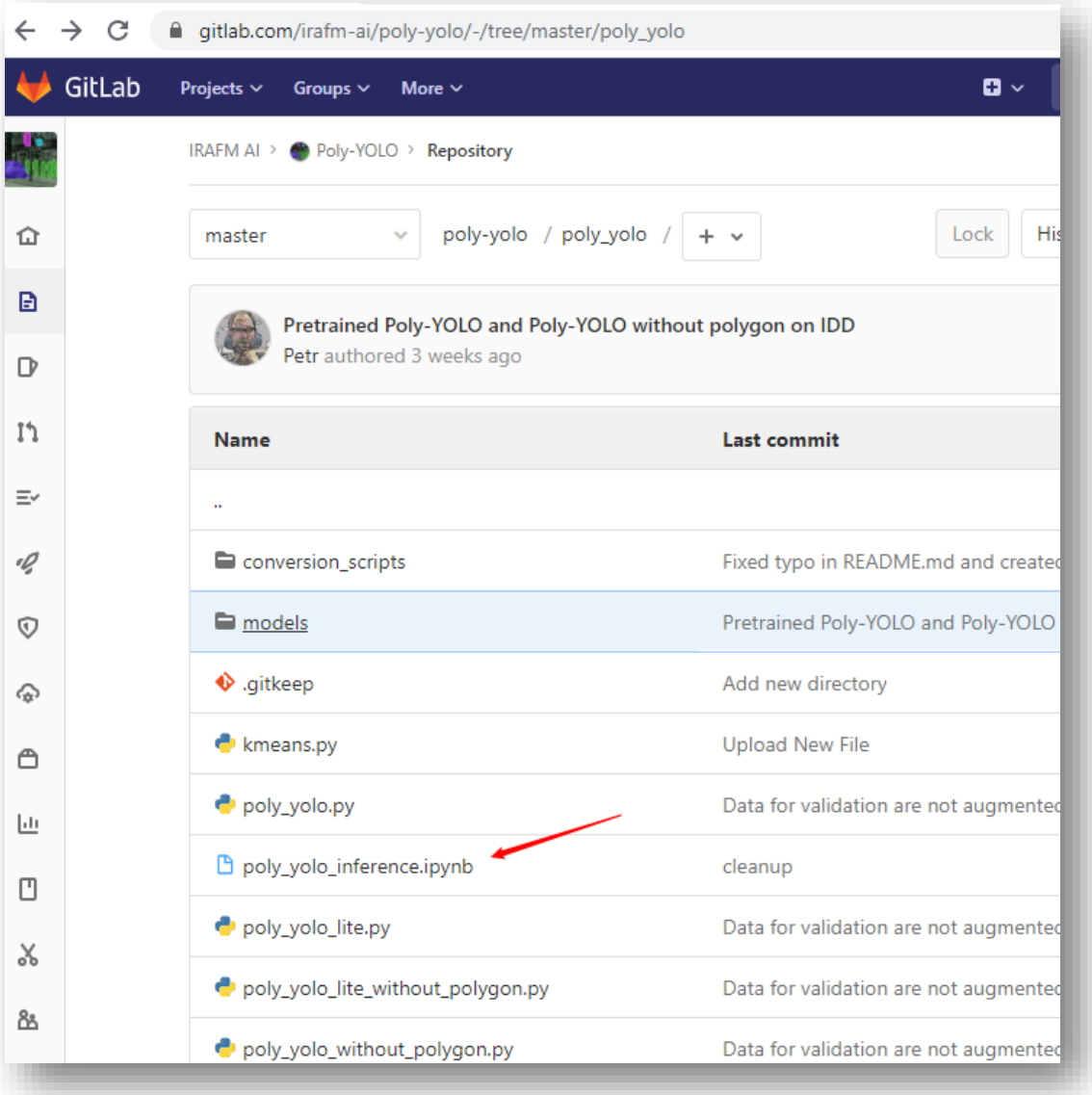
Change present working directory.

```
[2]   1 %cd /content/drive/My Drive/poly-yolo
```

```
/content/drive/My Drive/poly-yolo
```

```
  1 !git clone https://gitlab.com/irafm-ai/poly-yolo.git
```

3. Config paths of input data.



Copy the content of the inference.ipynb, cell by cell.

Modify the paths.

+ Code + Text

```
[ ]  21    if cls == 19: return (80, 80, 128)
```

```
1 dir_imgs_name = '/content/drive/My Drive/poly-yolo/poly_yolo/test_images/' #path_where_are_images_to_clasifi
2 out_path      = '/content/drive/My Drive/poly-yolo/poly_yolo/poly_yolo_predict/' #path, where the images wi
3 list_of_imgs = [root+"/"+name for root, dirs, files in os.walk(dir_imgs_name) for name in files]
4 list_of_imgs.sort()
5
6 #browse all images
7 total_boxes = 0
8 imgs        = 0
```

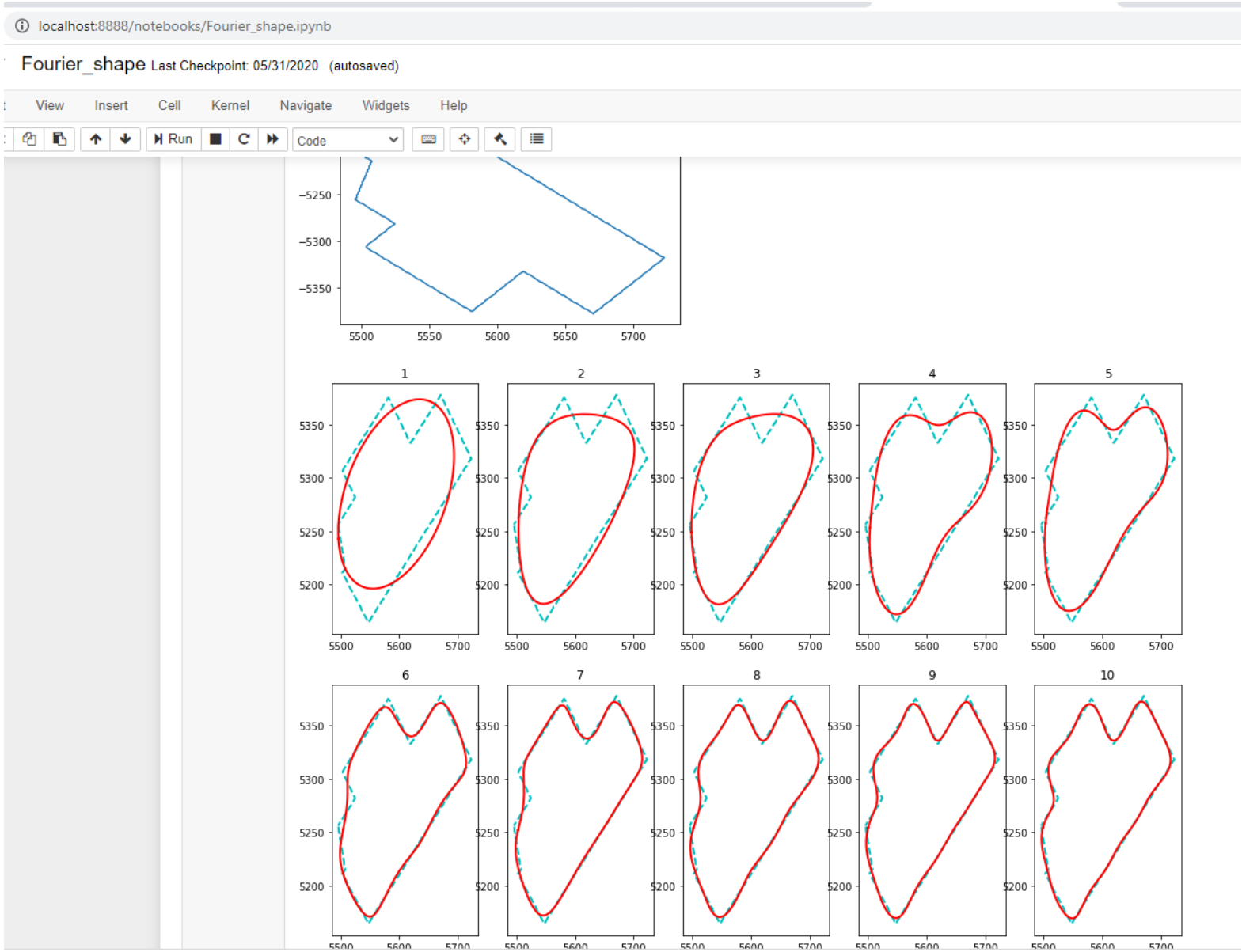
Tips:

1. If a tool does not fit you, use another one.

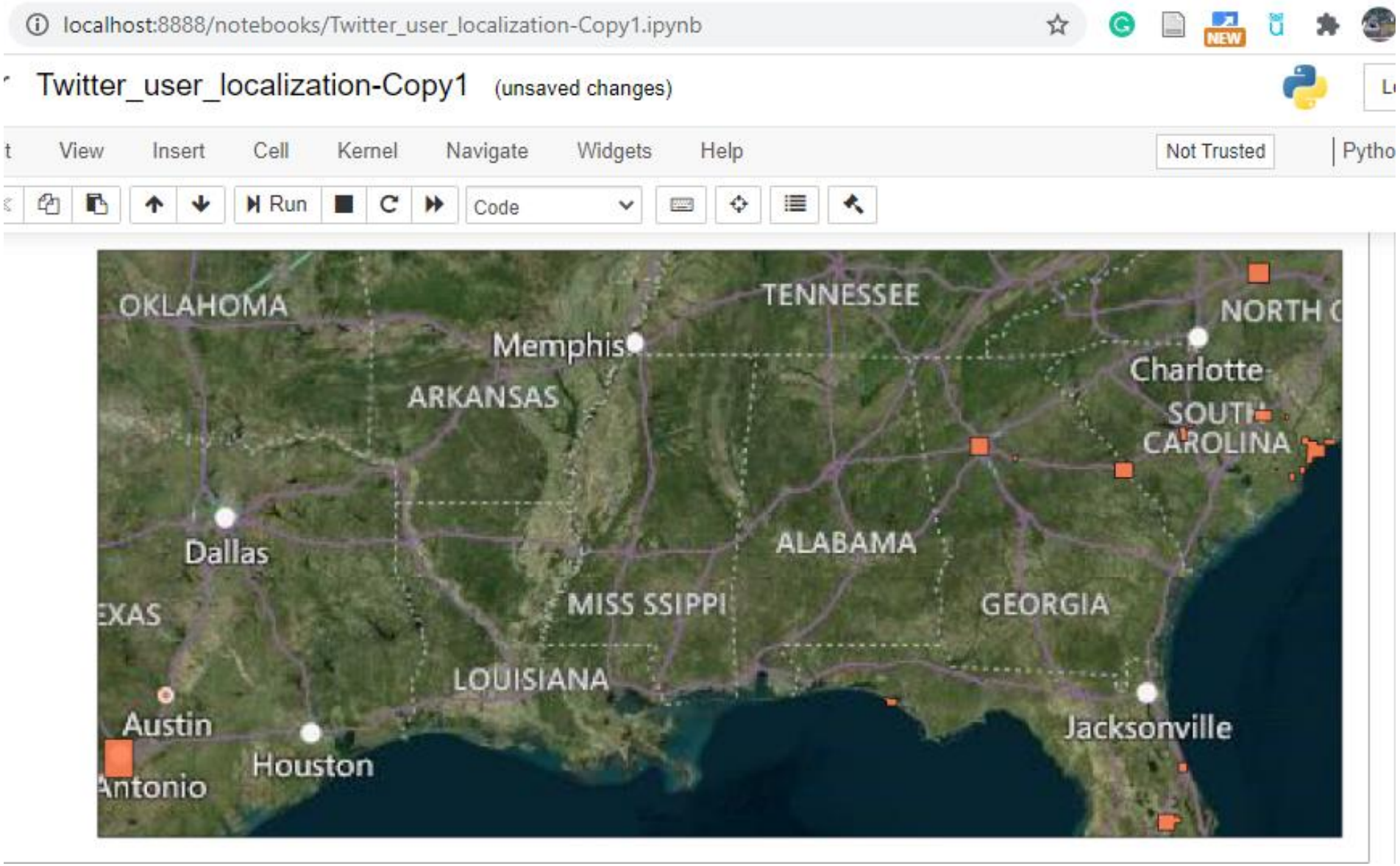
1. Colab: suitable for simple, small repository.

2. Jupyter notebook: good for pilot experiment (e.g. image/table processing), not good for multi-processing.

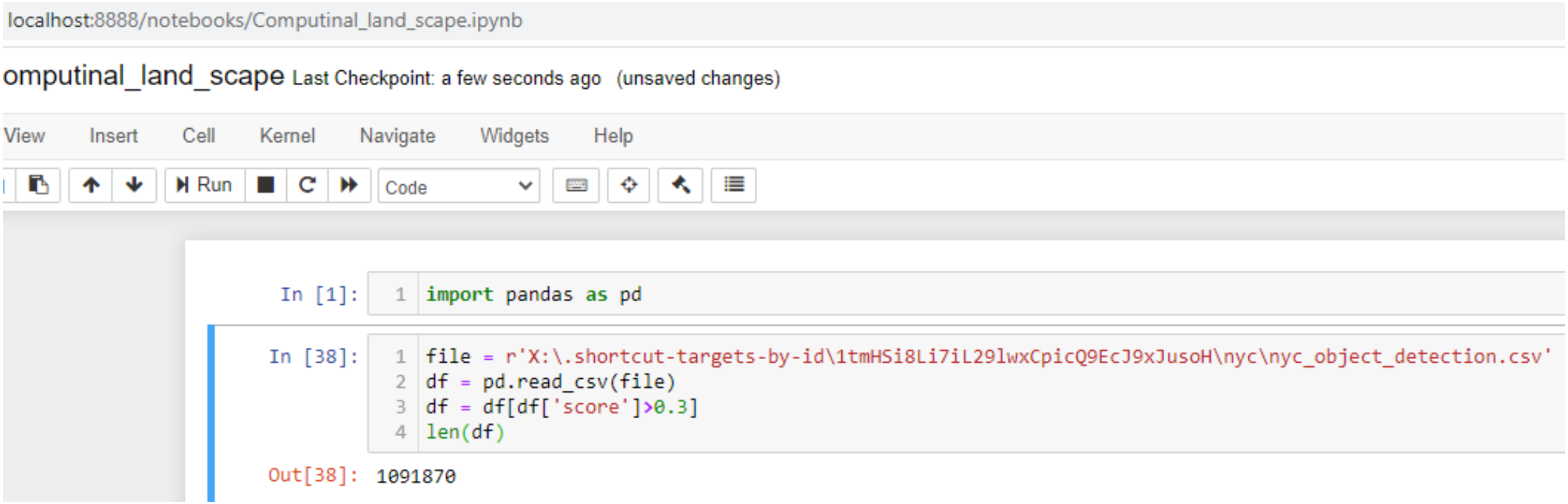
Example for Jupyter Notebook: shape analysis



Example for Jupyter Notebook: geographic analysis



Example for Jupyter Notebook: large tables



Question?