

به نام خدا

پروژه‌ی شبکه‌های عصبی

400443183

آرش محمد قلی نژاد

لینک کولب نهايى

لینک دیتاست رو بوفلو

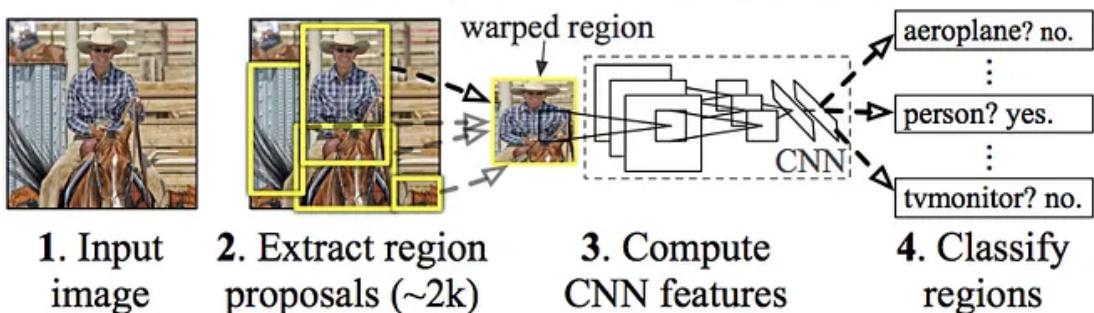
بخش اول توضیح تشخیص اشیا

در تسک های مربوط به پادگیری ماشین و شبکه های عصبی مفهومی به اسم **object detection** داریم که به ظاهر همان کلاسیفیکردن تصویر مبایش است اما تقاضت هایی دارند. اینکه در شناسایی اشیا شما علاوه بر اینکه کلاس آجکت درون تصویر را تشخیص میدهید موقعیت آن را یا ناحیه‌ی حضور آن را نیز تشخیص خواهد داد در واقع شناخت اشیا در تصویر از دو بخش کلاسیفیکیشن و لوکالایزیشن تشکیل شده است.

برای اینکار شبکه های عصبی مختلفی همچون RCNN ، FASTER RCNN ، YOLO از مدل‌های شبکه عصبی مرسوم مبایشند.

شبکه‌ی عصبی RCNN مخفف شده‌ی region based cnn مبایش است، در این شبکه ها از مفهومی به اسم **proposal** ها استفاده می‌شود تا به وسیله‌ی آنها اجکت های درون تصویر را بیابند.

R-CNN: Regions with CNN features



ابتدا تعدادی **region proposal** با استفاده از الگوریتم هایی مانند selective search می‌باشیم. سپس یک شبکه‌ی عصبی کانون‌لوشنی عمل کلاسیفیکیشن را انجام میدهد

الگوریتم سلکتیو سرج نیز یکی از الگوریتم های ساخت این ناحیه ها می‌باشد که با توجه به اینکه Intensity پیکسل های تصاویر چقدر مبایش آنها را با یکدیگر گروه‌بندی می‌کند در مقاله‌ی RCNN نویسنده‌گان 2000 ناحیه را انتخاب کردند.

تابع ارزیابی Loss نیز ترکیبی از تابع ارزیابی برای کلاسیفیکردن و ارزیابی regression مبایش است.

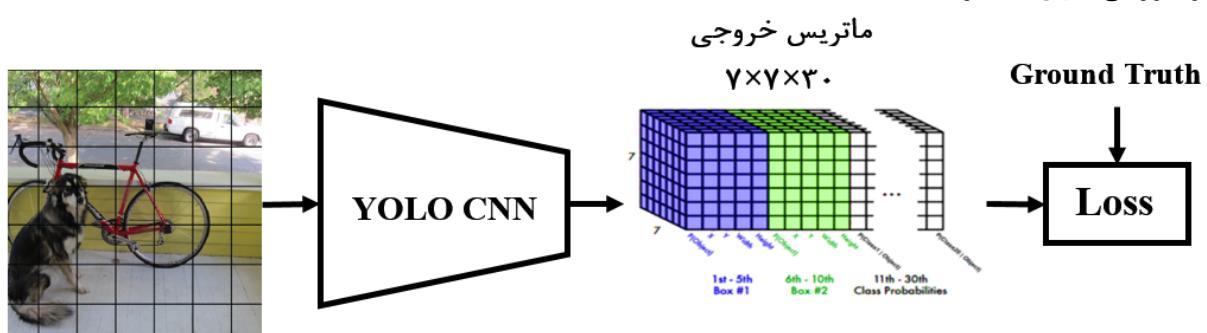
بخش رگرسیون مربوط به **bounding box** ها مبایش در حقیقت ما میخواهیم هم به درستی کلاس بندی کنیم هم اینکه به درستی مثل ناحیه‌ی حاوی یک بادکنک را تشخیص دهیم.



اما نکته این هست که rcnn الگوریتم کنید میباشد و به همین خاطر fast rcnn , faster rcnn و you only look once yolo یا ko-چکتری قطعه قطعه میشود معمولاً به صورت جدول شکل و قطعات برابر این اتفاق میافتد.

در یولو تنها یک شبکه کانولوشنی وجود دارد که تصویر ریسایز ورودی را دریافت (مرحله 1) و سپس به صورت همزمان چندین باکس را به همراه احتمال کلاس‌ها پیش‌بینی می‌کند (مرحله 2). YOLO ۲ روی تصاویر کامل آموزش می‌بیند و مستقیماً کارآئی تشخیص را بهبود می‌دهد.

ساختار کلی الگوریتم YOLO در شکل ۵ نشان داده شده است. تصویر ورودی با ابعاد $448 \times 448 \times 3$ به یک Grid یا شبکه $S \times S$ تقسیم‌بندی می‌شود. این تصویر به شبکه YOLO داده می‌شود. خروجی شبکه کانولوشنی، ماتریسی به ابعاد $S \times S \times 30$ خواهد بود. هریک از درایه‌های ماتریس $S \times S$ خروجی معادل با یک سلوول در شبکه $S \times S$ ورودی است (به ورودی و خروجی در شکل ۵ دقت کنید). خروجی $S \times S \times 30$ شامل مختصات باکس‌ها و احتمال‌هast است. اگر در فرآیند آموزش (Train) باشیم، خروجی $S \times S \times 30$ به همراه باکس‌های واقعی یا هدف (Ground Truth) به تابع اتلاف داده می‌شود. مقدار S در یولو نسخه ۱، برابر با ۷ در نظر گرفته شده است. اگر در فرآیند آزمایش (Test) باشیم، خروجی $S \times S \times 30$ به الگوریتم حذف غیرحداکثرها (Non-maximum Suppression) داده می‌شود تا باکس‌های ضعیف از بین بروند و تنها باکس‌های درست در خروجی نمایش داده شوند.



بخش دوم) تولید داده ها

من از دست خط خودم و 3 نفر دیگر استقاده کردم داده ها را به دو صورت تولید کردم صفحاتی که که فقط 5 کاراکتر اعداد فرد 1 تا 5 و مثبت و منفی بوده اند و دومین شکل نوشتن معادلات و برچسب زدن خود معادله همراه با کاراکتر هاست در زیر عکس هایی برای نمونه گذاشته شده اند.

$$\begin{array}{ccccccc}
 & & & \Rightarrow 1+1-1 \\
 1 & 1 & 1 & 1 & 1 & 1 & \\
 & & & \Rightarrow 3 - 5 - 1 & \Rightarrow 1-3+5 \\
 1 & 1 & 1 & 1 & 1 & 1 & \\
 & & & \Rightarrow 3 - 3 - 3 & \Rightarrow 1+4 \\
 1 & 1 & 1 & 1 & 1 & 1 & \\
 & & & \Rightarrow 3 - 5 - 1 & \\
 1 & 1 & 1 & 1 & 1 & 1 & \\
 & & & \Rightarrow 3 - 1 - 1 & \Rightarrow 1-3 \\
 1 & 1 & 1 & 5 & 5 & 5 & \\
 & & & \Rightarrow 5 - 3 - 1 & \Rightarrow 1-3+5 \\
 5 & 5 & 5 & 5 & 5 & 5 & \\
 & & & \Rightarrow 1-3-5-5 & \Rightarrow 1-3-3 \\
 5 & 5 & 5 & 5 & 5 & 5 & \\
 & & & \Rightarrow 1-3+3+5 & \\
 3 & 3 & 3 & 3 & 3 & 3 & \\
 & & & \Rightarrow 1-3-3-5 & \Rightarrow 5+5+5 \\
 3 & 3 & 3 & 3 & 3 & 3 & \\
 & + & + & + & + & + & \\
 + & - & - & - & - & - &
 \end{array}$$

$$\begin{array}{ccccc}
 5 & & 3 & & \\
 3 & & + & & \\
 5 & + & 1 & + & 3 \\
 1 & 3 & - & 5 & \\
 5 & - & 3 & & \\
 5 & & 5 & &
 \end{array}$$

$$\begin{aligned}
 &\Rightarrow 3 - 5 + 5 \\
 &\Rightarrow 1 + 1 - 5 + 5 \\
 &\Rightarrow 1 - 1 + 5 - 5 - 5
 \end{aligned}$$

$$\begin{aligned}
 &\Rightarrow 3 + 3 - 5 \\
 &\Rightarrow 3 - 3 - 5 \\
 &\Rightarrow 3 - 3 - 5 \\
 &\Rightarrow 3 - 3 - 5 \\
 &\Rightarrow 5 - 1 - 1 \\
 &\Rightarrow 1 - 1 - 1 \\
 &\Rightarrow 3 - 5 - 5 \\
 &\Rightarrow 5 - 3 - 3 \\
 &\Rightarrow 1 - 1 - 1 \\
 &\Rightarrow 1 + 1 + 1
 \end{aligned}$$

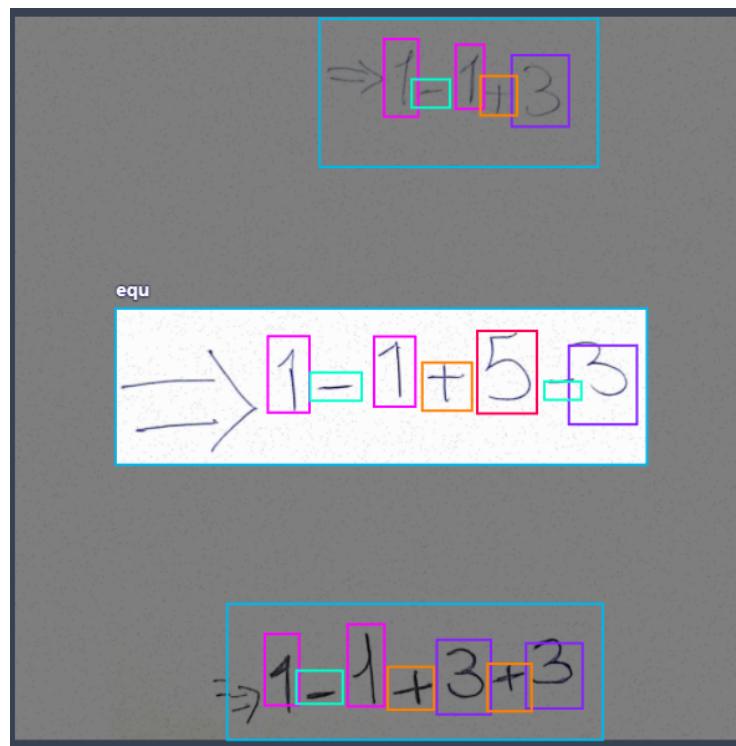
$$\begin{aligned}
 &\Rightarrow 3 + 3 - 5 \\
 &\Rightarrow 1 - 1 + 5 - 5 \\
 &\Rightarrow 1 - 1 + 5 - 5
 \end{aligned}$$

در
کل

عکس ها مشابه همین نمونه ها میباشند.

سایت رو بولو سایپی برای ساخت دیتا است در این سایت تصاویر را لیبل میزنیم و سپس باید از دیتا آگمنتیشن استقاده کنیم.
 الف) دیتا آگمنتیشن استقاده اش این هست که برای بیشتر کردن داده ها و همچنین جلوگیری از بیش برآذش و کمک به مدل برای تشخیص بهتر آجکت ها میباشد اینطوری مدل میتواند بهتر فعالیت کند و به ویژگی های مهم تر تصاویر توجه نماید و در مجموع GENERALIZATION انجام دهد.

ب) در زیر نمونه ای از تصویر لیبل خورده در دیتا است آورده شده اما برای بررسی تصاویر دیگر از این [لينک](#) میتوانید دیتا است را دانلود کنید.



ج) در این مرحله دیتا را پیش پردازش میکنیم و داده هارا با آگمنتیشن برای مدل آماده میکنیم. فقط برای پری پرسس از تغییر سایز استفاده کردم نسبت تصویر را تغییر ندادم فقط بهش های سفیدی به گوشه های تصویر اضافه شده تا همه ی تصاویر نسبت و تعداد پیکسل یکسان داشته باشند.

3 Preprocessing

② What can preprocessing do?

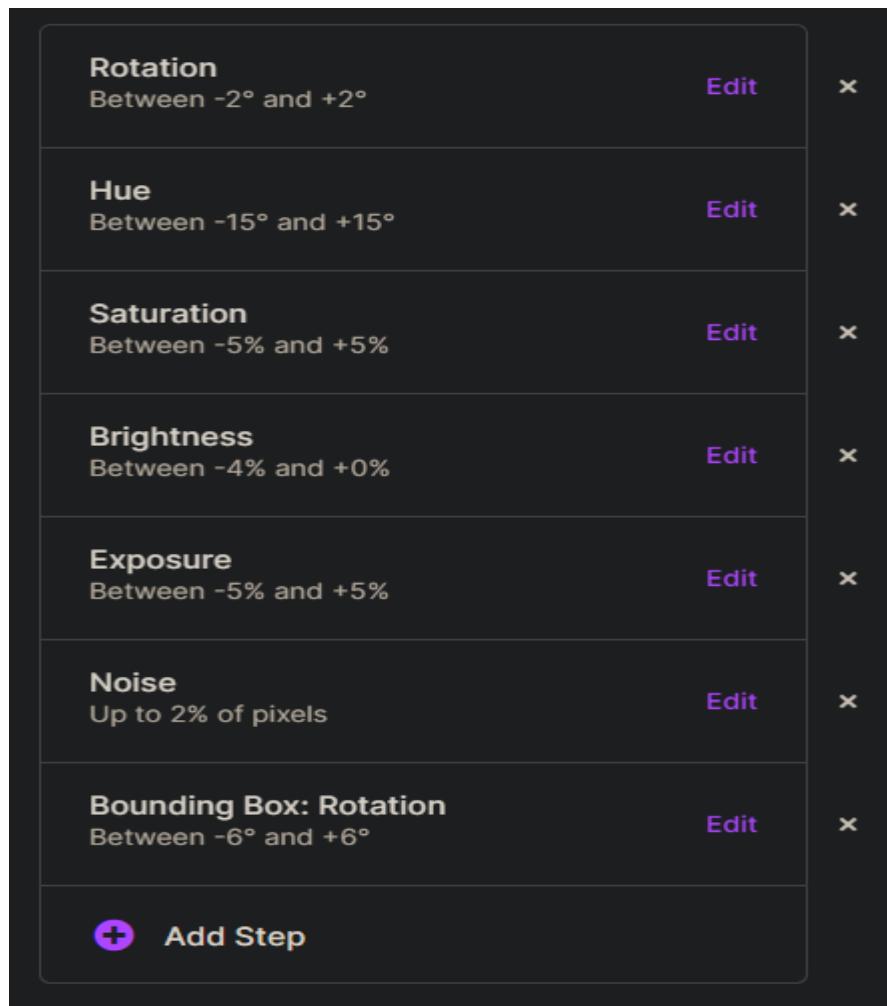
Decrease training time and increase performance by applying image transformations to all images in this dataset.

Resize
Fit (white edges) in 2048×2048 Edit ×

+ Add Step

Continue

لیست آگمنتیشن ها را در زیر مشاهده مینمایید.
همچنین در مرحله ۲ تصویر برای تست و تعدادی برای اموزش و ۴ عدد تصویر برای داده های ولیدیشن استفاده کردم



د) دلیل استفاده از آگمنتیشن روتویشن به خاطر اینکه همیشه کاراکتر ها را صاف روی کاغذ نمیبینیم. هیو و سچوریشن و برایتنس و اکسپوزر همگی دلیل مشترکی دارند که ما تصویر را از دیوایس های مختلف دریافت میکنیم این تصاویر رو کاغذ های مختلف هستند و نوشته ها ممکن است کمرنگ تر یا پر رنگ تر باشند که نیاز است مدل ما از پیش برای همچین حالاتی تعیین بیند. نویز به این خاطر که خط خورده‌گی یا هر چیزی را بتواند از تصویر اصلی آبجکت تقسیک کند. چرخش مستطیل های لیبل ها به خاطر اینکه قاعده نیازمند این هستیم که داده ها را از جهات مختلف بینیم مثلًا زاویه دار باشد. ولی ممکن است خود معادله جهت متغیری داشته باشد.

بخش سوم)

الف) لینک که در توضیحات دیناست آمده ولی در کد هم با این دستور فایل را دانلود و در کولب اکسترکت میکنیم

```
!curl -L "https://app.roboflow.com/ds/sI18Cc5T4J?key=WPV1CGrKUF" > roboflow.zip; unzip roboflow.zip; rm roboflow.zip
```

مدل دیتکتران 2 را با دستورات زیر نصب میکنیم در کنارش نسخه‌ی پایتورچ مناسب آن و ... را نیز نصب میکنیم

```
# install detectron2:
!pip install detectron2==0.1.3 -f https://dl.fbaipublicfiles.com/detectron2/wheels/cu101/torch1.5/index.html
```

```
# install dependencies: (use cu101 because colab has CUDA 10.1)
!pip install -U torch==1.5 torchvision==0.6 -f https://download.pytorch.org/whl/cu101/torch_stable.html
!pip install cython pyyaml==5.1
!pip install -U 'git+https://github.com/cocodataset/cocoapi.git#subdirectory=PythonAPI'
import torch, torchvision
print(torch.__version__, torch.cuda.is_available())
!gcc --version
# opencv is pre-installed on colab
```

در ادامه باید دیتاست را رجیستر نماییم

```
from detectron2.data.datasets import register_coco_instances
register_coco_instances("my_dataset_train", {}, "/content/train/_annotations.coco.json", "/content/train")
register_coco_instances("my_dataset_val", {}, "/content/valid/_annotations.coco.json", "/content/valid")
register_coco_instances("my_dataset_test", {}, "/content/test/_annotations.coco.json", "/content/test")
```

دیتاست فقط یه سری تصویر نیست و فرمت خاصی دارد از جمله مختصات جعبه ها و پس برای نمایش دادن به صورت یک نمونه از Visualizer دیتکران استفاده میکنیم.

```
for d in random.sample(dataset_dicts, 1):
    img = cv2.imread(d["file_name"])
    visualizer = Visualizer(img[:, :, ::-1], metadata=my_dataset_train_metadata, scale=0.2)
    vis = visualizer.draw_dataset_dict(d)
    cv2.imshow(vis.get_image()[:, :, ::-1])
```

این کد به صورت رندوم یک نمونه انتخاب میکند و نمایش میدهد.

در اینجا ما کانفیگوریشن یا تنظیمات مدل هنگام آموزش را لاحظ میکنیم و در کنار آن trainer میسازیم تا مدل را آموزش دهیم تعداد اپیک را 1500 تعیین کردیم البته مینیمم و مаксیمم تعیین میکنیم گاما و لرنینگ ریت را به ترتیب 0.05 و 0.01 تعیین میکنیم و تعداد کلاس ها را تعیین میکنیم از مدل faster_rcnn استفاده مینماییم و سپس آموزش آغاز میشود.

```
from detectron2.config import get_cfg
import os

cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/faster_rcnn_X_101_32x8d_FPN_3x.yaml"))
cfg.DATASETS.TRAIN = ("my_dataset_train",)
cfg.DATASETS.TEST = ("my_dataset_val",)

cfg.DATALOADER.NUM_WORKERS = 4
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-Detection/faster_rcnn_X_101_32x8d_FPN_3x.yaml")
cfg.SOLVER.IMS_PER_BATCH = 4
cfg.SOLVER.BASE_LR = 0.01

cfg.SOLVER.WARMUP_ITERS = 1200
cfg.SOLVER.MAX_ITER = 1500
cfg.SOLVER.STEPS = (1200, 1500)
cfg.SOLVER.GAMMA = 0.05

cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 64
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 7

cfg.TEST.EVAL_PERIOD = 50

os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
trainer = CocoTrainer(cfg)
```

چون توی کولب طی چند روز مدل رو ترین کردم و تست کردم و کرو تکمیل کردم و وزن ها رو ساخته بودم از قبل به همین خاطر توی لینک کولب و فایل نهایی دقت ها نیست اینجا همشون رو توی انتهای تمرین اوردم در واقع از ورژن آخر تمرین داده شده ی مدل اون هم توی قسمت revision های کولب هست لینکش هم برآتون میزارم.

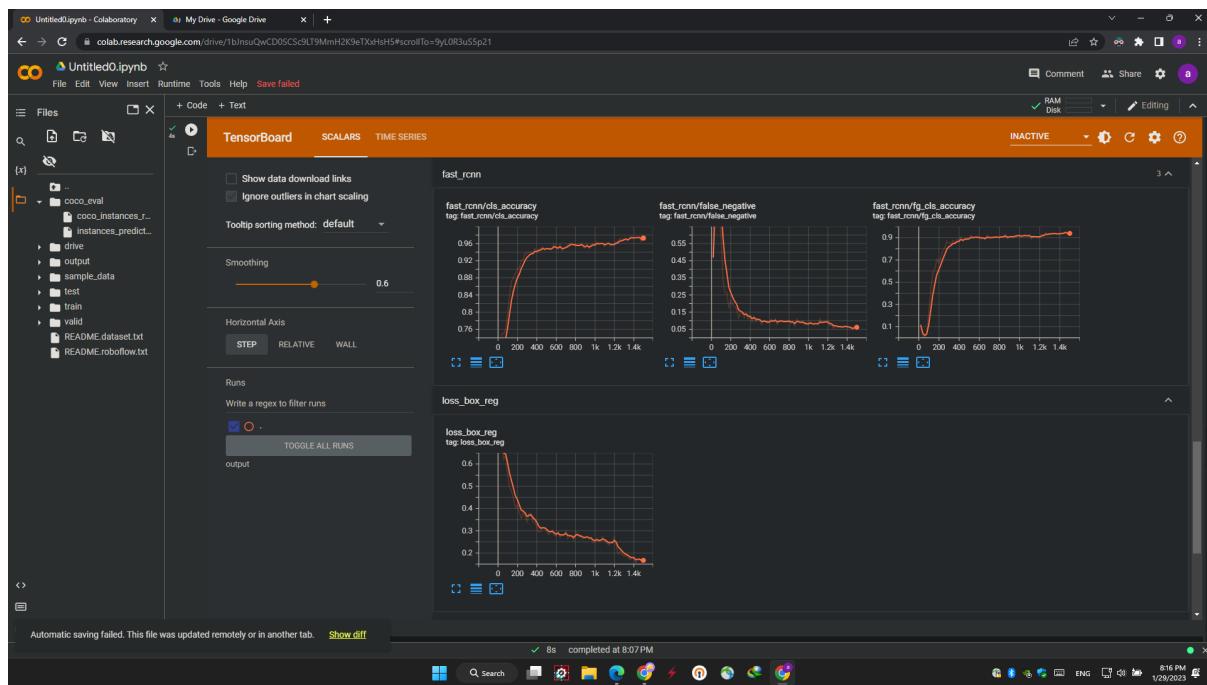
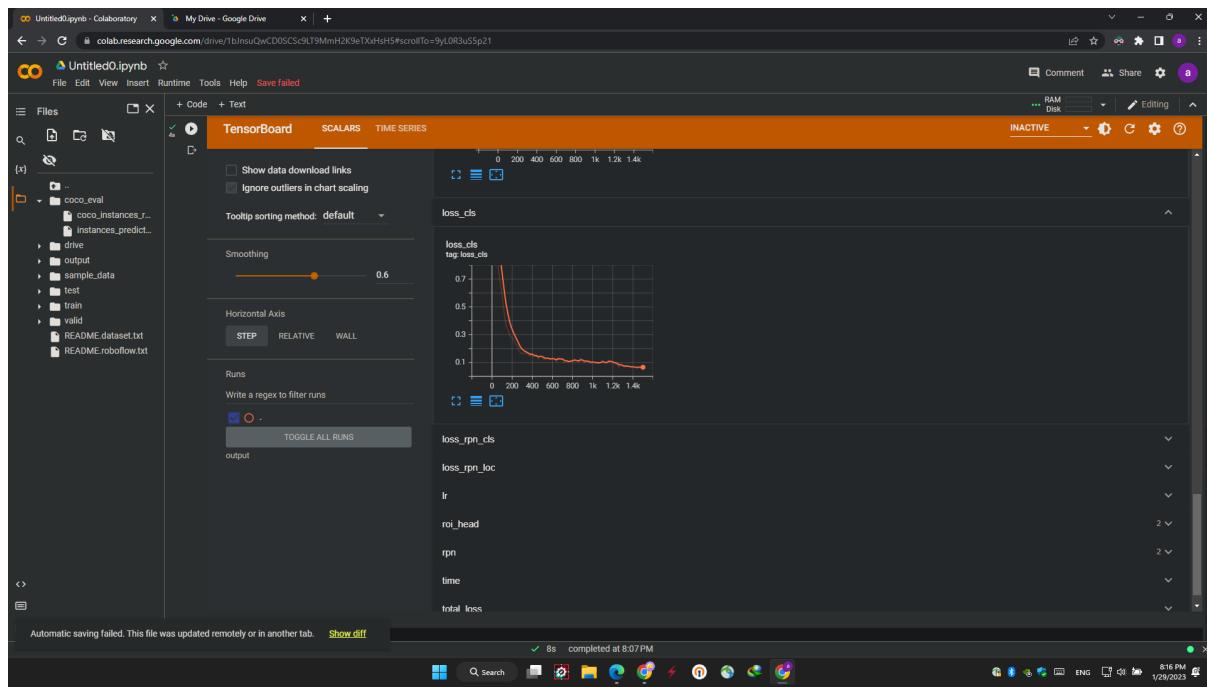
```

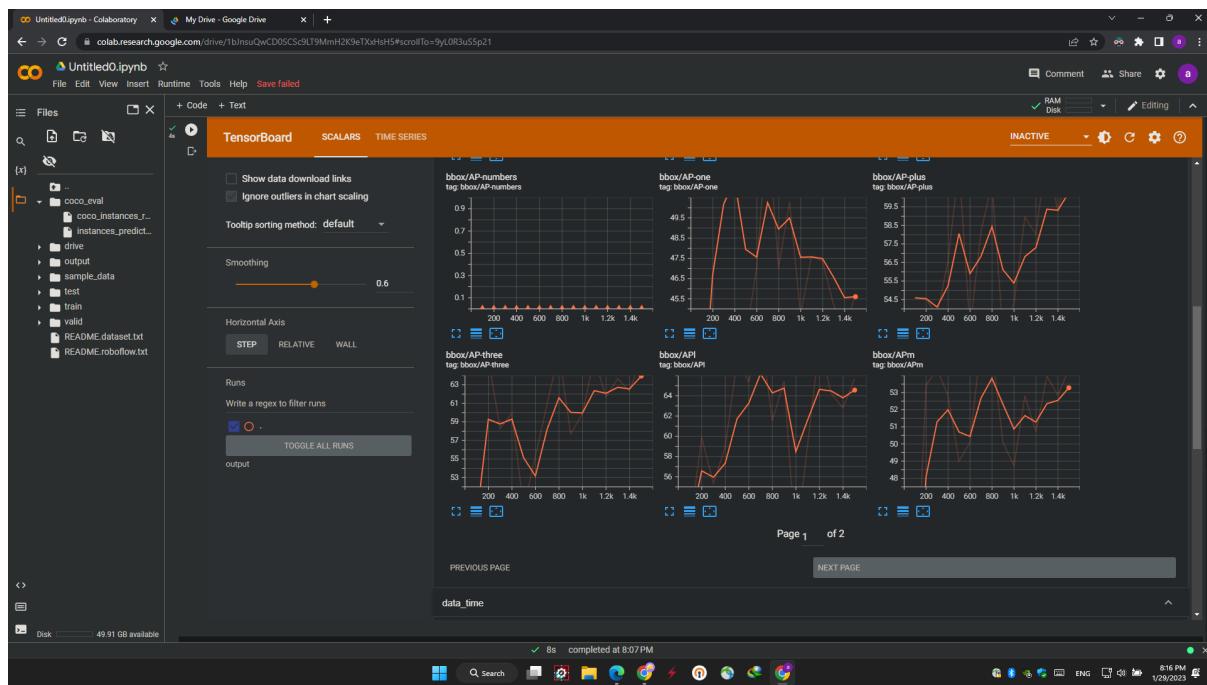
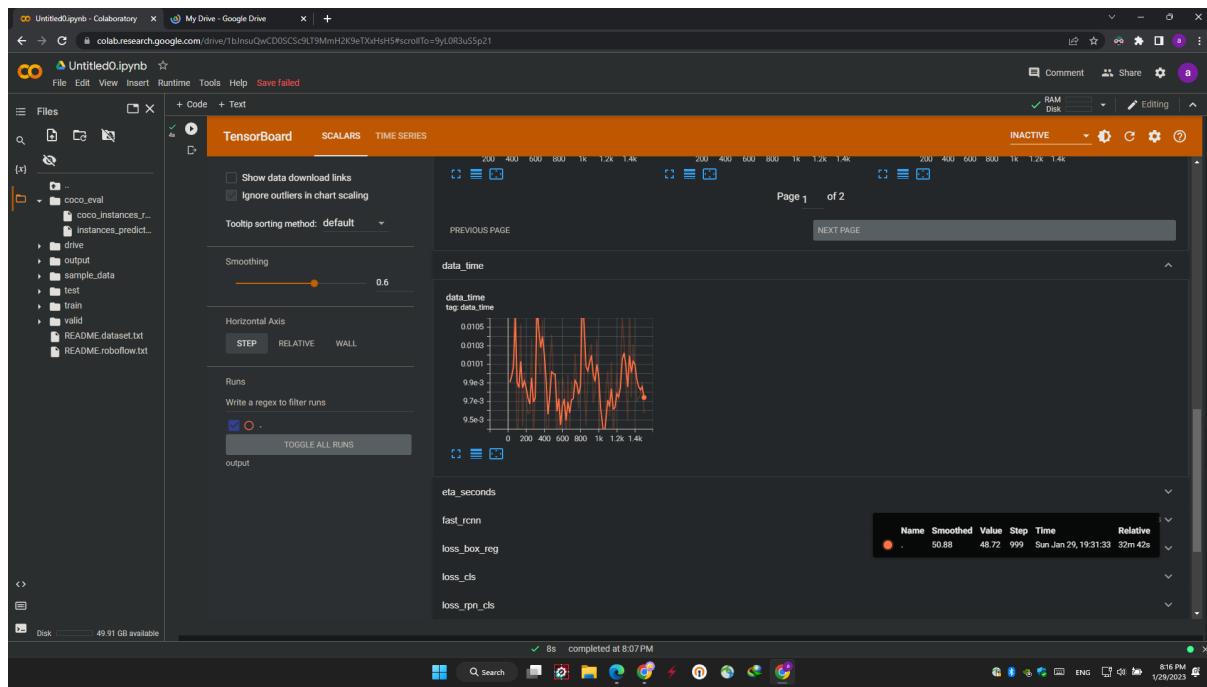
Average Precision (AP) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.541
Average Precision (AP) @[ IoU=0.50 | area= all | maxDets=100 ] = 0.981
Average Precision (AP) @[ IoU=0.75 | area= all | maxDets=100 ] = 0.560
Average Precision (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
Average Precision (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.544
Average Precision (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.657
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 1 ] = 0.075
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 10 ] = 0.515
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.598
Average Recall (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
Average Recall (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.594
Average Recall (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.683
[01/29 16:19:47 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
| AP      | AP50    | AP75    | APs     | AP1     | |
|---|---|---|---|---|---|
| 54.062 | 98.124 | 55.972 | nan     | 54.389 | 65.706 |
[01/29 16:19:47 d2.evaluation.coco_evaluation]: Note that some metrics cannot be computed.
[01/29 16:19:47 d2.evaluation.coco_evaluation]: Per-category bbox AP:
| category | AP      | category | AP      | category | AP      |
|:-----:|:-----:|:-----:|:-----:|:-----:|:-----:|
| numbers  | nan    | equ     | nan    | five    | 58.989 |
| minus    | 37.924 | one     | 45.676 | plus    | 61.711 |
| three    | 66.012 |          |          |          |          |
[01/29 16:19:47 d2.engine.defaults]: Evaluation results for my_dataset_val in csv format:
[01/29 16:19:47 d2.evaluation.testing]: copypaste: Task: bbox
[01/29 16:19:47 d2.evaluation.testing]: copypaste: AP,AP50,AP75,APs,APm,AP1
[01/29 16:19:47 d2.evaluation.testing]: copypaste: 54.0624,98.1240,55.9723,nan,54.3894,65.7058
[01/29 16:19:47 d2.utils.events]: eta: 0:00:02 iter: 1499 total_loss: 0.301 loss_cls: 0.066 loss_box_reg: 0.166
[01/29 16:19:47 d2.engine.hooks]: Overall training speed: 1497 iterations in 0:53:36 (2.1490 s / it)
[01/29 16:19:47 d2.engine.hooks]: Total training time: 0:54:29 (0:00:52 on hooks)

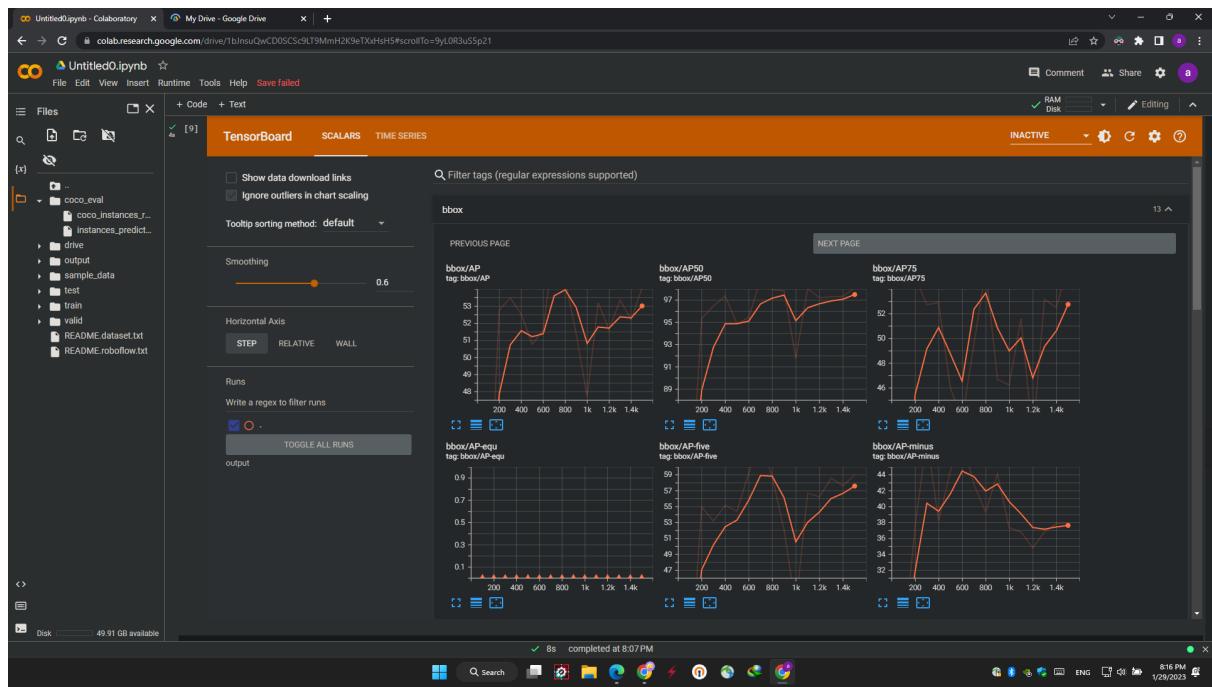
loss_box_reg: 0.166 loss_rpn_cls: 0.004 loss_rpn_loc: 0.060 time: 2.1475 data_time: 0.0096 lr: 0.000500 max_mem: 7203M

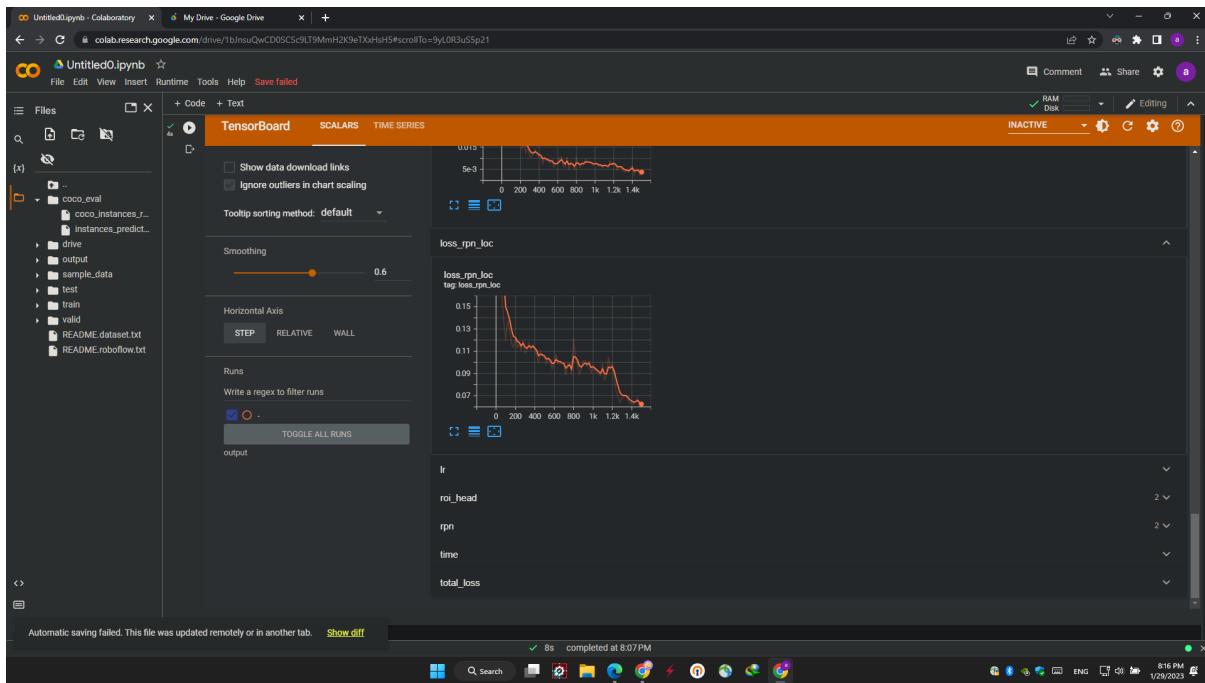
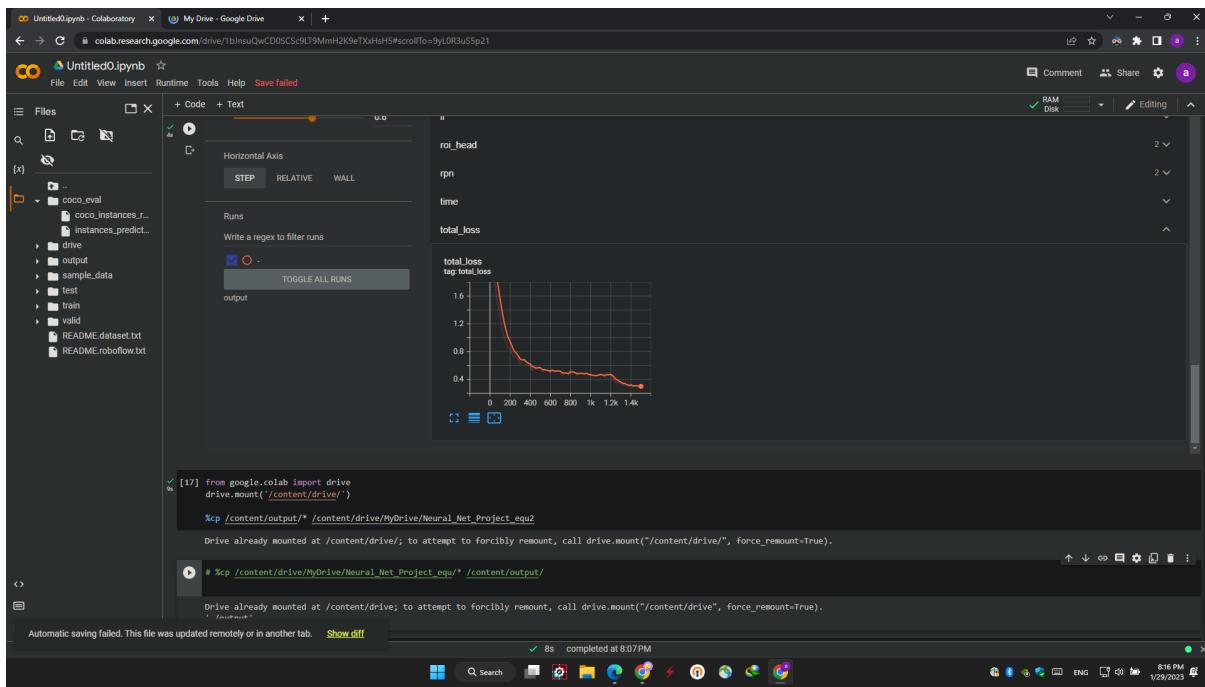
```

Average precision ها و لاس تجمعی نهایی به این صورت هست.
 این [لينک](#) کد نهایی نیست لینک بالا اول گزارش و درون فایل زیپ نهایی هست ولی چون یک قسمت رو مجبور بودم دفعات بعدی اجرا کنم تمرینات توی اون قسمت نبودن خروجی اینجا پرینت شده هست.
 تنسور بوردها نیز در لینک اول گزارش صفحه‌ی نخست لحاظ شده و قابل مشاهده میباشد که البته من تعدادی از نمودار مهم‌ها رو اینجا در این تصویر آورده‌ام.





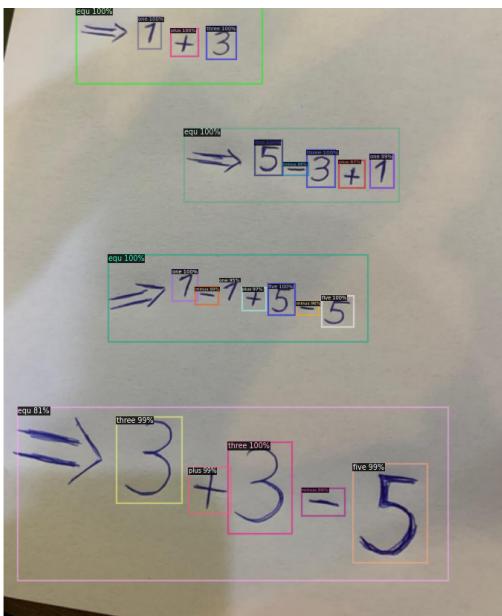




ولى توى لينكە دقىقىرش هىت.

```
%cp /content/drive/MyDrive/output/* /content/output/
%cp /content/drive/MyDrive/coco_eval/* /content/coco_eval/
```

وزن هارا با كد بالا از فolder `output` مېفستىم توى فolder دىگر تنسور بوردم همین فolderه ولى يە فolder دىگم من فرستادم.
در مرحله ى بعد هم داده ى تست را ازمایش مىكئيم.
تصاویر تست من به دو تصویر زير خلاصه مىشوند.



```

for imageName in glob.glob('/content/test/*jpg'):
    im = cv2.imread(imageName)
    outputs = predictor(im)
    v = Visualizer(im[:, :, ::-1],
                   metadata=test_metadata,
                   scale=0.4
                  )
    out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
    cv2.imshow(out.get_image()[:, :, ::-1])

```

تکه کد بالا داده های تست را ویژوآلایز مینماید

```

cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "model_final.pth")
cfg.DATASETS.TEST = ("my_dataset_test", )
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.8 # set the testing threshold for this model
predictor = DefaultPredictor(cfg)
test_metadata = MetadataCatalog.get("my_dataset_test")

```

در این قسمت نیز پریدیکتور را با کانفیگ مشخص ساختیم من تصمیم گرفتم 0.8 را برای حد آستانه در نظر بگیرم یعنی 80 درصد اطمینان حد پایین برای تشخیص و لیبل زدن آجکت باشد.

```

Instances(num_instances=24, image_height=2048, image_width=2048, fields=[pred_boxes: Boxes(tensor([[ 531.5068,  831.2450, 1404.6719,
1123.6541],
[ 935.3122, 1460.9542, 1150.3837, 1768.6832],
[ 633.0607, 32.0995, 710.2997, 138.4920],
[1023.8582, 446.7924, 1117.8896, 562.9416],
[ 747.2914, 879.9861, 831.4532, 987.2453],
[ 787.2723, 406.8260, 1510.5396, 652.7413],
[1070.5510, 929.3508, 1159.3297, 1034.6917],
[ 742.9150, 71.4713, 835.0174, 164.4297],
[1199.4156, 479.8297, 1294.1339, 603.4644],
[1249.8970, 968.4471, 1358.1595, 1073.6636],
[ 424.5056, 0.0000, 1050.2268, 255.5228],
[ 861.4347, 64.2730, 963.3210, 173.6006],
[ 801.7204, 1546.3751, 944.2788, 1701.4755],
[1411.2704, 494.0201, 1491.3755, 607.2872],
[ 560.0438, 1374.9254, 779.0957, 1666.6764],
[1183.5563, 1615.6753, 1328.1017, 1708.7352],
[1355.2405, 1532.4802, 1605.1816, 1866.7462],
[ 825.0922, 939.3187, 902.9456, 997.4454],
[1120.4800, 520.6005, 1205.9247, 564.1337],
[ 983.5591, 940.7565, 1062.2175, 1018.9290],
[1163.8523, 988.4493, 1240.0582, 1032.5841],
[ 905.9714, 909.4827, 978.1345, 998.6254],
[1387.5044, 512.9594, 1395.5436, 607.0536],
[ 228.7347, 1343.6530, 1675.0736, 1926.4214]], device='cuda:0')), scores: tensor([0.9986, 0.9985, 0.9983, 0.9975, 0.9970, 0.9969,
0.9966, 0.9963, 0.9962,
0.9960, 0.9953, 0.9951, 0.9949, 0.9946, 0.9938, 0.9934, 0.9929, 0.9878,
0.9767, 0.9739, 0.9649, 0.9526, 0.8706, 0.8121], device='cuda:0'), pred_classes: tensor([1, 6, 4, 2, 4, 1, 2, 5, 6, 2, 1, 6, 5, 4,
6, 3, 2, 3, 5, 3, 4, 5, 1],
device='cuda:0'))]

```

خروجی به شکل زیر میباشد یک سری مختصات حاوی دو نقطه میباشد این نقاط را میتوان استفاده کرد تا مستطیل ساخت.
در ادامه score ها و در انتها کلاس ها را داریم البته خود کلاس را نداریم به جایش شماره ای به کلاس نسبت داده آن را
داریم.

در ادامه لیست لیبل ها را میخواهیم که با کد زیر از MetaData ها آن را بدست میآوریم که درون آن دیکشنری ای جاوی
ایندکس لیبل ها قرار دارد. با کد زیر لیبل هر ایندکس را بدست میآوریم.

```

| labels_list = list(MetadataCatalog.get(cfg.DATASETS.TRAIN[0]).thing_classes)
|
| labels_num = list(MetadataCatalog.get(cfg.DATASETS.TRAIN[0]).thing_dataset_id_to_contiguous_id.values())
|
| labels_list
|
| ['numbers', 'equ', 'five', 'minus', 'one', 'plus', 'three']
|
| labels_num
|
| [0, 1, 2, 3, 4, 5, 6]

```

خود مدل چубه ها و لیبل هایش ترتیب خاصی ندارد.
روش تشخیص کاراکتر های درون معادله به شرح زیر میباشد:
ابتدا ما باکس هایی که برچسبشان معادله میباشد را با برミداریم همراهش نقاط مختصات مستطیل ها را نیز نامپای میکنیم.

```

pred_boxes = outputs['instances'].pred_boxes.tensor.cpu().numpy()
equ = outputs['instances'][outputs['instances'].pred_classes==labels_num[labels_list.index('equ')]].pred_boxes.tensor.cpu().numpy()

```

در ادامه میخواهیم بررسی کنیم درون مستطیل یک معادله چند تا کاراکتر و چیا داریم ممکن است مستطیل کاراکتر کمی بیرون
بزند برای همین با متدهای get_centers مراکز مستطیل ها را برミداریم.

```

pred_boxes_centers=list(pred_boxes_centers)
equ_characters=[]
for i in range(len(equ)):
    boxes_in_equ = []
    for j in range(len(pred_boxes_centers)):
        if pred_boxes_centers[j][0]>equ[i][0] and pred_boxes_centers[j][0]<equ[i][2] and pred_boxes_centers[j][1]>equ[i][1] and pred_boxes_centers[j][1]<equ[i][3]:
            boxes_in_equ.append(j)
    equ_characters.append(boxes_in_equ)
print(equ_characters)

```

با این کد ما کاراکتر هایی که درون هر مستطیل هستند را بدست می آوریم اینطوری اگر کاراکتر عجیب غریبی یه جای صفحه بود مدل به درستی ان را درون معادله چون نیست در نظر نمیگرفت.
در مرحله‌ی آخر لازم هست تا نقاط را بر اساس تقدیم دسته بندی کنیم با حلقه‌ی زیر این کار را انجام میدهیم.

```
equ_characters_x = []

for equ in equ_characters:
    eq1 = []
    for j in range(len(equ)):
        eq1.append(pred_boxes_x[equ[j]])
    equ_characters_x.append(eq1)
```

مختصات را مانند ایندکس نقطه‌ی درون هر معادله تعیین میکنیم مختصات محور اکس را در نظر میگیریم.

```
equ_labels=[]
for data in outputs["instances"].pred_classes:
    num=data.item()
    equ_labels.append(MetadataCatalog.get(cfg.DATASETS.TRAIN[0]).thing_classes[num])

for equ in equ_characters_x:
    for x in sorted(equ):
        j=equ.index(x)
        i=equ_characters_x.index(equ)
        print(equ_characters[i][j], end=" ")
print()
```

با این کار مرتب سازی میکنیم سپس بر اساس اینکه کدام لیبل اول امده میتوانیم ان را اول محاسبه کنیم.

```
sums=[]
for equ in equ_characters_x:
    sign = "+"
    sum = 0
    for x in sorted(equ):
        j=equ.index(x)
        i=equ_characters_x.index(equ)

        if equ_labels[equ_characters[i][j]]=="one":
            if sign=="+":
                sum += 1
            elif sign=="-":
                sum -= 1
        elif equ_labels[equ_characters[i][j]]=="three":
            if sign=="+":
                sum += 3
            elif sign=="-":
                sum -=3
        elif equ_labels[equ_characters[i][j]]=="five":
            if sign=="+":
                sum += 5
            elif sign=="-":
                sum -=5
        elif equ_labels[equ_characters[i][j]]=="minus":
            sign = "-"
        elif equ_labels[equ_characters[i][j]]=="plus":
            sign = "+"
        elif equ_labels[equ_characters[i][j]]=="equ":
            pass
    sums.append(sum)
```

در این مرحله میگوییم که اگر لبیل حروف بود حرف + - را ذخیره میکنیم ولی اگر به عدد رسیدیم بر اساس اینکه کدام علامت پیش از آن امده جمع یا تفریق مینماییم.

```
for i in range(len(equs)):

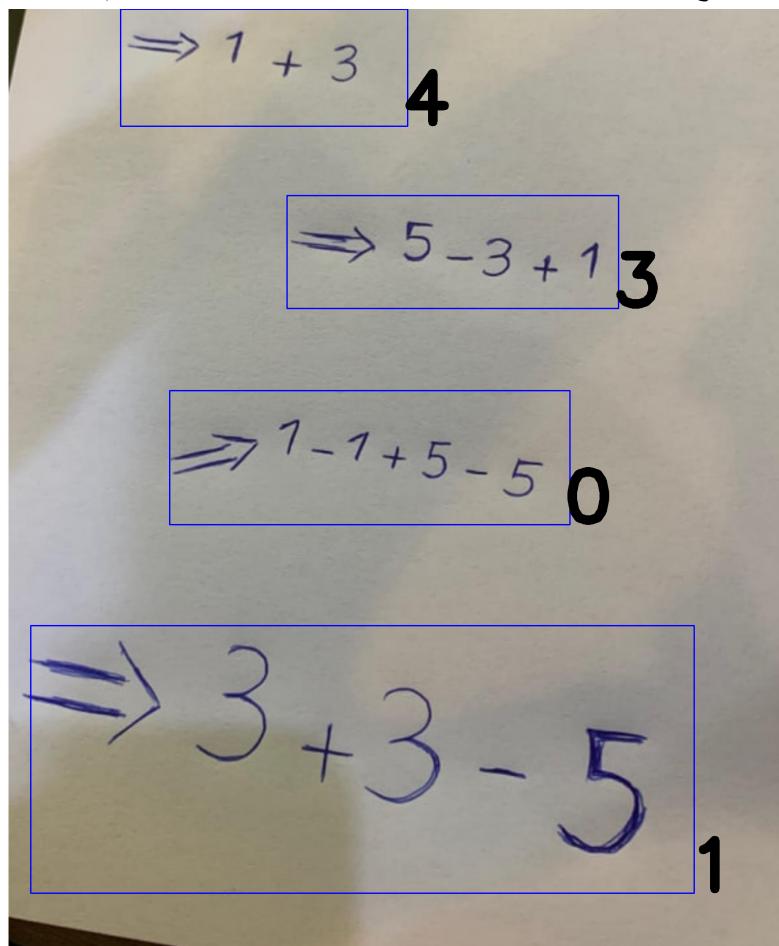
    start_point = (int(equs[i][0]),int(equs[i][1]))

    end_point = (int(equs[i][2]),int(equs[i][3]))

    sumsum = str(sums[i])
    color = (255, 0, 0)
    position = (int(equs[i][2])-10, int(equs[i][3])-10)
    thickness = 2
    font = cv2.FONT_ITALIC
    im = cv2.rectangle(im, start_point, end_point, color, thickness)
    im = cv2.putText(im, sumsum, position, font, 5, (0, 0, 0), 20)

cv2_imshow(im)
```

با تکه کد بالا نتایج را به ترتیب برای هر معادله در کنار مختصاتش ذخیره میکنیم، مساله حل شد.



خروجی نهایی برای تصویر معادلات به شکل مقابل میباشد.
هر 4 معادله به درستی تشخیص داده شد.

