

گزارش کار پروژه درس شبکه های عصبی

سید محمدرضا حسینی ۹۷۲۴۳۱۲۹

بیثا باروطیان ۴۰۱۴۴۳۰۱۹

ابخش صفر:

کارکرد CNN: از ویژگی های اصلی CNN میتوان به داشتن لایه کانولوشن-لایه pooling – لایه fully connected اشاره کرد. لایه های early تر به ویژگی های ساده تر مثل رنگ و لبه توجه میکنند در صورتی که لایه های عمیق تر به ویژگی ها و خصوصیات پیچیده تر میپردازند. لایه کانولوشن component هایی مانند دیتای ورودی فیلتر و feature map میباشد. فیلتر در فیلد های receptive تصویر حرکت داده میشود و چک میکند که آیا ویژگی موجود است یا نه که این عملیات در اصل همان کانولوشن است. لایه pooling: همان لایه down sampling که کاهش ابعاد را به همراه دارد و تعداد پارامتر های ورودی را میکاهد. لایه FC: انجام عملیات classification براساس ویژگی های استخراج شده در لایه های پیشین (برای این کار عموماً از softmax) استفاده میکند. CNN معماری های مختلفی دارد مانند ResNet-LeNet و ... به علاوه. CNN کاربرد های مختلفی دارد که میتوان به استفاده در مارکتینگ-پزشکی (استفاده در رادیولوژی برای تشخیص بیماری و ...) -e-commerce (پیشنهاد راهکار برای سوددهی در مدیا)-صنعت خودرو سازی (کمک به تشخیص خطوط مسیر و امنیت) اشاره کرد.

یادگیری transfer: در اصل تکنیکی است که مدل روی یک تسک train و روی تسک دیگر اجرا میشود. میتوانیم از دو روش این اینکار را انجام بدهیم. در روش اول در ابتدا تسک source را به صورتی انتخاب میکنیم که مدلی قابل پیش بینی برای آن وجود داشته باشد و ارتباطی بین دیتای ورودی-دیتای خروجی و ایده های train شده در حین نگاشت ورودی به خروجی وجود داشته باشد. سپس باید برای آن تسک یک مدل را به صورتی develop کنیم که یادگیری ویژگی ها را تضمین کند. در مرحله بعد از این مدل که develop شد میتوانیم از این مدل در تسک دیگری استفاده کنیم (به عنوان نقطه آغازی برای مدل تسک دومی) در این مرحله میتوان بر حسب انتخاب از کل مدل یا قسمتی از مدل استفاده کرد. در نهایت میتوان مدل را fine tune کرد به این صورت که classifier را در ConvNet جایگزین و دوباره آموزش دهیم و و همچنین وزن های شبکه از پیش آموزش دیده شده را از طریق back propagation تنظیم کنیم.

مدل های state of the art برای تشخیص اشیا:

Object detector ها ویژگی ها را از تصویر ورودی/فریم ویدیو استخراج می کنند. ابتدا اشیا (و جعبه های مرزی آنها) را پیدا می کنند سپس آنها را طبقه بندی می کنند. معمولا از یک backbone ایجاد شده اند. در حالت کلی دو نوع مدل object detector داریم یک مرحله ای و دو مرحله ای. معماری دو مرحله ای در ابتدا یک object region proposal میگیرد و سپس آن را بر اساس ویژگی های استخراج شده از منطقه پیشنهادی طبقه بندی می کند. دارای دقت بالا اما زمان بر هستند. پس برای کارهایی مثل تشخیص مانع در ماشین خودران مناسب نیستند. از نمونه های این نوع مدل ها میتوانیم به Fast-RCNN , MaskRCNN و RCNN اشاره کنیم. مدل یک مرحله ای یک مرحله ای جعبه مرزی را بر روی تصاویر بدون مرحله پیشنهاد منطقه پیش بینی می کند و به سرعت تشخیص بیشتر می رسد. از نمونه های این نوع مدل میتوانیم به YOLO-SSD و RetinaNet EfficientDet-DV اشاره کنیم.

منابع:

<https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-۳bd۲b۱۱۶۴a۵۳>

<https://www.ibm.com/topics/convolutional-neural-networks>

<https://medium.com/@pedroazevedo۱/object-detection-state-of-the-art-۲۰۲۲-ad۷۵۰e۰f۱۰۰۳>

<https://machinelearningmastery.com/transfer-learning-for-deep-learning/>

<https://medium.com/deeplearningsandbox/how-to-use-transfer-learning-and-fine-tuning-in-keras-and-tensorflow-to-build-an-image-recognition-۹۴b۰b۰۲۴۴۴f۲>

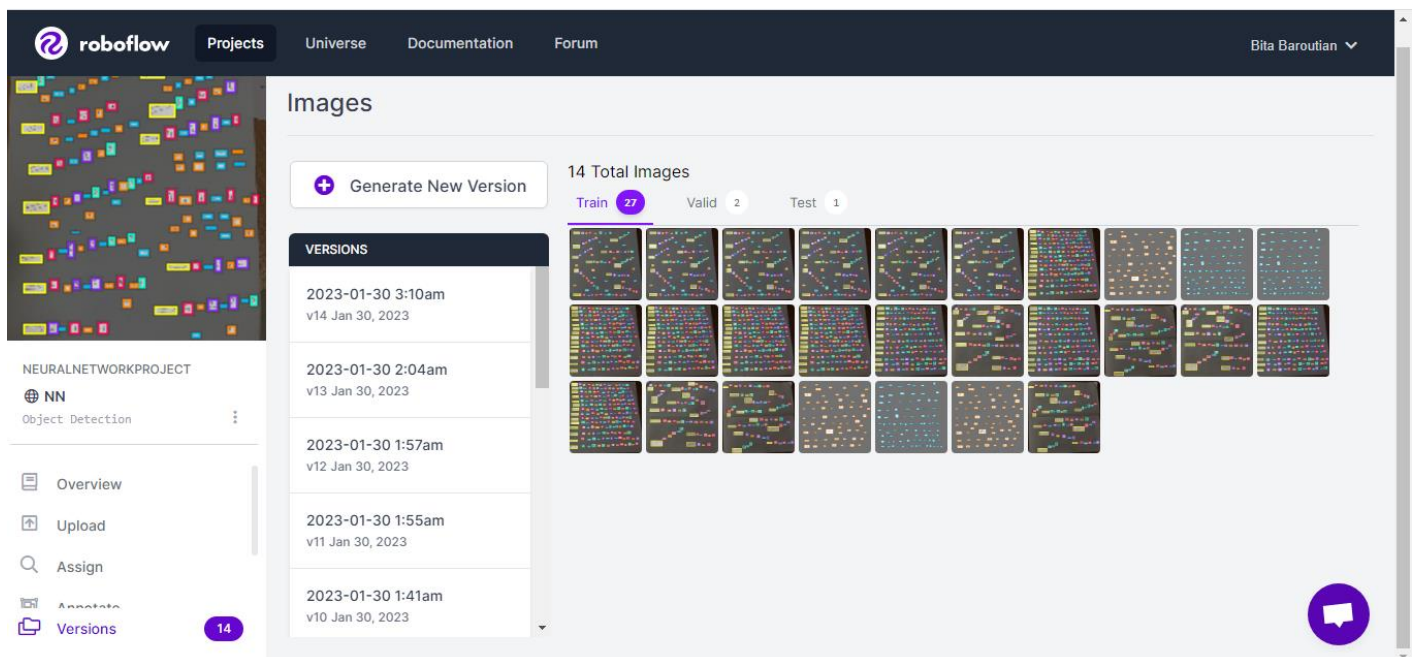
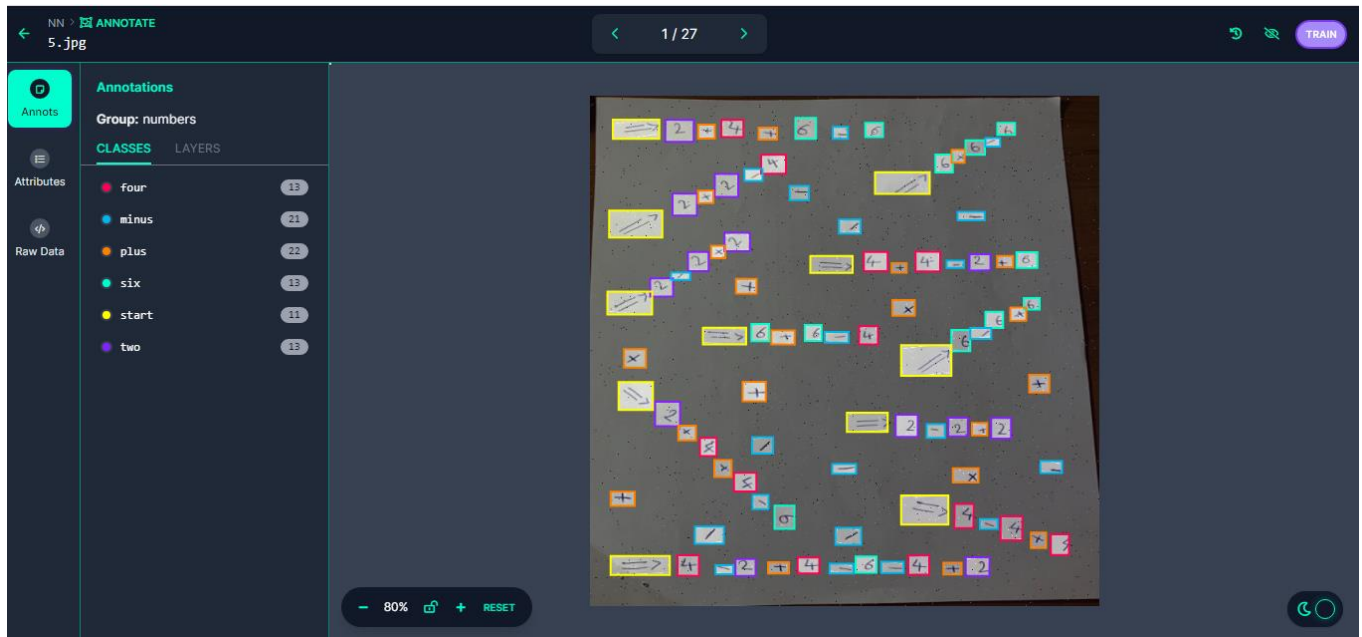
<https://stats.stackexchange.com/questions/۳۴۳۷۶۳/fine-tuning-vs-transferlearning-vs-learning-from-scratch#:~:text=Transfer%۲۰learning%۲۰is%۲۰when%۲۰a,the%۲۰model%۲۰with%۲۰a%۲۰dataset>

<https://medium.com/@pedroazevedo۱/object-detection-state-of-the-art-۲۰۲۲-ad۷۵۰e۰f۱۰۰۳>

۲-بخش یک و دو :

نمونه ای از دست خط و لیبل زدن آنها با roboflow:

دارای ۶ کلاس: two-four-six-start-plus-minus:



ویژگی های مجموعه تصاویر:

در مسائل Object Detection با توجه به اینکه مدل باید بتواند از یک آبجکت در یک نقطه از تصویر را در میان تعداد زیادی آبجکت تشخیص بدهد. برای اینکه بتواند این کار را انجام دهد باید تعداد مناسب و قابل قبولی از انواع آبجکت هایی که باید تشخیص دهد مشاهده کند و روی آن ها آموزش ببیند. در صورتی که این تعداد داده بخواهد توسط نویسنده سیستم ایجاد شود مدت زمان زیادی وقت را باید فقط برای ساخت دیتاست صرف کند. با استفاده از Data augmentation میتوان با داشتن تعداد کمی داده میتوان یک دیتاست بزرگ ایجاد کرد که در آن نمونه های آموزشی با توجه به تکنیک های به کار رفته تفاوت دارند و میتواند جنرالیزیشن بهتری ایجاد کند.

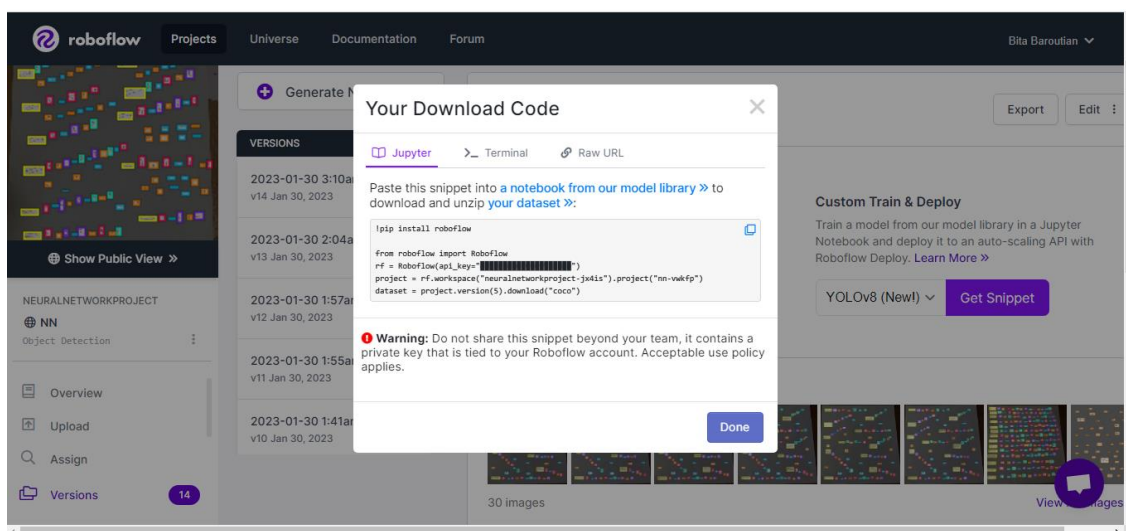
پس از اینکه دیتاست اول را ایجاد کردیم و چندین مدل مختلف را با آن آموزش دادیم به این نتیجه رسیدیم که شاید داده های به خوبی نوشته نشده باشند به همین علت دیتاست دیگری نیز ایجاد کردیم. بعد از آن با استفاده از تکنیک های مختلف Data augmentation چندین دیتاست مختلف از این دو مجموعه داده ایجاد کردیم که در آن هر نمونه تفاوت با دیگری داشته باشد. سپس مدل را با استفاده از تمامی این دیتاست ها آموزش دادیم.

PREPROCESSING	Auto-Orient: Applied Resize: Stretch to 1000×1000	PREPROCESSING	Auto-Orient: Applied Resize: Stretch to 640×640
AUGMENTATIONS	Outputs per training example: 3 Rotation: Between -10° and +10° Shear: ±2° Horizontal, ±2° Vertical Brightness: Between -30% and +30% Bounding Box: Rotation: Between -2° and +2° Bounding Box: Noise: Up to 1% of pixels	AUGMENTATIONS	Outputs per training example: 3 Blur: Up to 0.75px Noise: Up to 1% of pixels Bounding Box: Rotation: Between -10° and +10° Bounding Box: Brightness: Between -10% and +10%
DETAILS	Version Name: 2023-01-30 12:39am Version ID: 9 Generated: Jan 30, 2023 Annotation Group: numbers	DETAILS	Version Name: 2023-01-29 12:13pm Version ID: 5 Generated: Jan 29, 2023 Annotation Group: numbers

از چندین تکنیک Data augmentation برای ایجاد دیتاست ها استفاده شده است. از blur برای تار کردن تصویر استفاده شده تا در مواقعی که تصویر مقدار تار است باز هم بتواند درست تشخیص دهد. سپس نویز اضافه کردیم (یک درصد) تا در صورت وجود نویز salt and pepper هم مدل ما بتواند کارکرد خود را حفظ کند. از دو نوع روتیشن استفاده کرده ایم. یک مدل روتیشن تک تک باکس ها و دیگری روتیشن کامل تصویر. با این کار در صورتی که آبجکت های نوشته شده مقدار زاویه دار باشد یا اینکه تصویر مقدار چرخیده باشد باز نیز مدل خواهد توانست تشخیص دهد آن ها را. از shear نیز استفاده شده است تا در صورتی که عکس گرفته شده زاویه دار بوده و از بالا گرفته نشده باشد در تشخیص به مشکل بر نخورد. از brightness و exposure نیز استفاده شده تا در صورتی که مقدار تصویر به علت نور کم یا مشکلات دوربین این مشکلات را داشت مدل نسبت به آن مقاوم باشد.

در ابتدا یک

سپس با استفاده از این کد مدل را به گوگل کولب منتقل میکنیم.



لینک دیتاست ها:

[Project Overview \(roboflow.com\)](https://roboflow.com/project/overview)

[Number Dataset > Overview \(roboflow.com\)](https://roboflow.com/dataset/number-dataset/overview)

۳- بخش سوم:

در ابتدا dependency های مورد نیاز را نصب و import میکنیم.

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

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Looking in links: https://download.pytorch.org/whl/cu101/torch_stable.html
Collecting torch==1.5
  Downloading https://download.pytorch.org/whl/cu101/torch-1.5.0%2Bcu101-cp38-cp38-linux_x86_64.whl (703.8 MB)
    703.8/703.8 MB 2.1 MB/s eta 0:00:00
Collecting torchvision==0.6
  Downloading https://download.pytorch.org/whl/cu101/torchvision-0.6.0%2Bcu101-cp38-cp38-linux_x86_64.whl (6.6 MB)
    6.6/6.6 MB 60.6 MB/s eta 0:00:00
Requirement already satisfied: future in /usr/local/lib/python3.8/dist-packages (from torch==1.5) (0.16.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.8/dist-packages (from torch==1.5) (1.21.6)
Requirement already satisfied: pillow>=4.1.1 in /usr/local/lib/python3.8/dist-packages (from torchvision==0.6) (7.1.2)
Installing collected packages: torch, torchvision
  Attempting uninstall: torch
    Found existing installation: torch 1.13.1+cu116
    Uninstalling torch-1.13.1+cu116:
      Successfully uninstalled torch-1.13.1+cu116
  Attempting uninstall: torchvision
```

```
+ Code + Text
Requirement already satisfied: matplotlib>=2.1.0 in /usr/local/lib/python3.8/dist-packages (from pycocotools==2.0) (3.2.2)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.8/dist-packages (from matplotlib>=2.1.0->pycocotools==2.0) (1.4.4)

import detectron2
from detectron2.utils.logger import setup_logger
setup_logger()

# import some common libraries
import numpy as np
import cv2
import random
from google.colab.patches import cv2_imshow

# import some common detectron2 utilities
from detectron2 import model_zoo
from detectron2.engine import DefaultPredictor
from detectron2.config import get_cfg
from detectron2.utils.visualizer import Visualizer
from detectron2.data import MetadataCatalog
from detectron2.data.catalog import DatasetCatalog
```

لینکی که روبوفلو داد را به گوگل کولب منتقل میکنیم: و instance مجموعه ها را رجیستر میکنیم

```
+ Code + Text
!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="4U1veSGmQ70eQ9V8rnHD")
project = rf.workspace("neuralnetworkproject-jx4is").project("nn-vukfp")
dataset = project.version(8).download("coco")

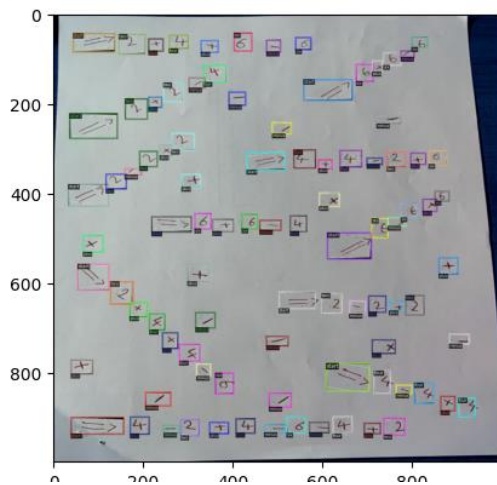
from detectron2.data.datasets import register_coco_instances
register_coco_instances("my_dataset_train_NN_8", {}, "/content/NN-8/train/ annotations.coco.json", "/content/NN-8/train")
register_coco_instances("my_dataset_val_NN_8", {}, "/content/NN-8/valid/ annotations.coco.json", "/content/NN-8/valid")
register_coco_instances("my_dataset_test_NN_8", {}, "/content/NN-8/test/ annotations.coco.json", "/content/NN-8/test")
```

نمایش نمونه ای از مجموعه train :

```
[ ] my_dataset_train_metadata = MetadataCatalog.get("my_dataset_train")
    dataset_dicts = DatasetCatalog.get("my_dataset_train")
    from matplotlib import pyplot as plt
    import random
    from detectron2.utils.visualizer import Visualizer
    window_name = 'image'
    for d in dataset_dicts:
        img = cv2.imread(d["file_name"])
        visualizer = Visualizer(img[:, :, ::-1], metadata=my_dataset_train_metadata, scale=1)
        vis = visualizer.draw_dataset_dict(d)
        plt.imshow(vis.get_image()[:, :, ::-1])
```

WARNING [01/30 03:12:11 d2.data.datasets.coco]:
Category ids in annotations are not in [1, #categories]! We'll apply a mapping for you.

[01/30 03:12:11 d2.data.datasets.coco]: Loaded 33 images in COCO format from ./NN-14/train/_annotations.coco.json



آموزش مدل با استفاده از detectron :

```
+ Code + Text
Connect Editing

from detectron2.engine import DefaultTrainer
from detectron2.evaluation import COCOEvaluator

class CocoTrainer(DefaultTrainer):

    @classmethod
    def build_evaluator(cls, cfg, dataset_name, output_folder=None):

        if output_folder is None:
            os.makedirs("coco_eval", exist_ok=True)
            output_folder = "coco_eval"

        return COCOEvaluator(dataset_name, cfg, False, output_folder)
```

Model zoo:

بسیاری از مدل های از پیش آموزش دیده را می توان در "modelzoo" یافت. این مجموعه ای از مدل های از پیش آموزش داده شده بر روی یک مجموعه داده خاص است که آماده استفاده است. اکثرا از وزن های از پیش آموزش دیده این مدل برای راه اندازی مدل سفارشی خود استفاده می شود. این روش به طور قابل توجهی زمان را کاهش و عملکرد را بهبود میبخشد.

نرخ یادگیری = ۰.۰۰۷ و تعداد iteration ها = ۱۰۰۰

```
cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/faster_rcnn_R_101_FPN_3x.yaml"))
# cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/retinanet_R_50_FPN_3x.yaml"))

cfg.DATASETS.TRAIN = ("my_dataset_train",)
cfg.DATASETS.TEST = ("my_dataset_val",)

cfg.DATALOADER.NUM_WORKERS = 1
# cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-Detection/faster_rcnn_R_101_FPN_3x.yaml") # Let training initialize from model zoo
# cfg.MODEL.WEIGHTS = "detectron2://ImageNetPretrained/MSRA/R-50.pkl"
cfg.MODEL.WEIGHTS = "output/model_final.pth"
cfg.SOLVER.IMS_PER_BATCH = 4
cfg.SOLVER.BASE_LR = 0.007

cfg.SOLVER.WARMUP_ITERS = 500
cfg.SOLVER.MAX_ITER = 1000 #adjust up if val mAP is still rising, adjust down if overfit
cfg.SOLVER.STEPS = (300, 600)
cfg.SOLVER.GAMMA = 0.005

cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 64
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 7 #your number of classes + 1

cfg.TEST.EVAL_PERIOD = 500

os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
trainer = CocoTrainer(cfg)
trainer.resume_or_load(resume=False)
trainer.train()
```

خروجی :


```

+ Code + Text
[01/30 03:23:34 d2.evaluation.fast_eval_api]: Accumulating evaluation results...
[01/30 03:23:34 d2.evaluation.fast_eval_api]: COCOeval_opt.accumulate() finished in 0.01 seconds.
Average Precision (AP) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.182
Average Precision (AP) @[ IoU=0.50 | area= all | maxDets=100 ] = 0.601
Average Precision (AP) @[ IoU=0.75 | area= all | maxDets=100 ] = 0.033
Average Precision (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.111
Average Precision (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.193
Average Precision (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 1 ] = 0.012
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 10 ] = 0.164
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.241
Average Recall (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.296
Average Recall (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.217
Average Recall (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
[01/30 03:23:34 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
| AP | AP50 | AP75 | APs | APm | AP1 |
|-----|-----|-----|-----|-----|-----|
| 18.189 | 60.088 | 3.258 | 11.120 | 19.347 | nan |
[01/30 03:23:34 d2.evaluation.coco_evaluation]: Some metrics cannot be computed and is shown as NaN.
[01/30 03:23:34 d2.evaluation.coco_evaluation]: Per-category bbox AP:
| category | AP | category | AP | category | AP |
|-----|-----|-----|-----|-----|-----|
| numbers | nan | four | 16.639 | minus | 5.845 |
| plus | 11.249 | six | 21.600 | start | 31.752 |
| two | 22.052 | | | | |
[01/30 03:23:34 d2.engine.defaults]: Evaluation results for my_dataset_val in csv format:
[01/30 03:23:34 d2.evaluation.testing]: copypaste: Task: bbox
[01/30 03:23:34 d2.evaluation.testing]: copypaste: AP, AP50, AP75, APs, APm, AP1
[01/30 03:23:34 d2.evaluation.testing]: copypaste: 18.1894, 60.0876, 3.2585, 11.1198, 19.3466, nan

```

معیار ها (COCO Evaluator): می‌تواند AP را برای instance segmentation-keypoint
 detection-box detection ارزیابی کند پس از آن، ما از build_detection_test_loader استفاده می‌کنیم که یک torch DataLoader را برمی‌گرداند، که مجموعه داده تشخیص داده شده را بارگیری می‌کند.

Average precision, recall : یعنی precision میانگین که:

$$Precision = \frac{TP}{TP + FP}$$

$$AP@{\alpha} = \int_0^1 p(r) dr$$

$$Recall = \frac{TP}{TP + FN}$$

در هنگام رسم منحنی precision recall که در آستانه IoU ارزیابی شده دقت متوسط (average precision) را بدست می‌آوریم. IoU تقسیم بین ناحیه همپوشانی و ناحیه اشتراک را ارزیابی می‌کند. به عبارت دیگر، میزان همپوشانی بین (gt) ground truth و پیش‌بینی‌ها (pd) را ارزیابی می‌کند. از ۰ تا ۱ متغیر است، جایی که ۱ یک همپوشانی کامل

بین حقیقت اصلی و پیش بینی است. در رابطه سمت راست منظور از a میزان آستانه p میزان $precision$ و r میزان $recall$ است.

فاز evaluation :

از کلاس DefaultPredictor استفاده می کنیم. البته ما از همان `cfg` که در طول آموزش استفاده کردیم استفاده خواهیم کرد. و دو پارامتر را برای استنتاج خود تغییر خواهیم داد.

```
[ ] from detectron2.utils.visualizer import ColorMode
import glob

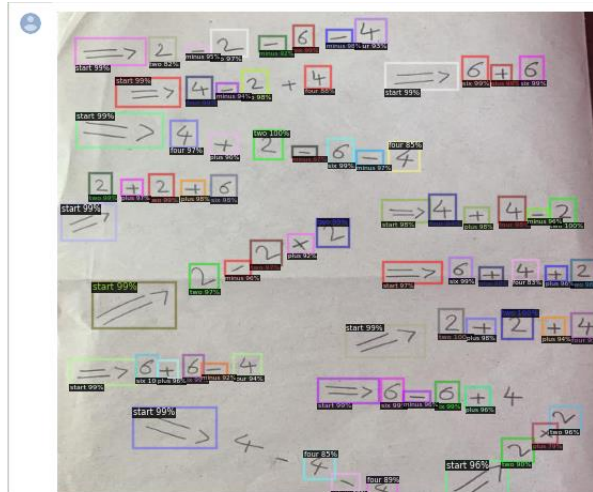
for imageName in glob.glob('/content/Number_finder-5/train/*.jpg'):
    im = cv2.imread(imageName)
    outputs = predictor(im)
    v = Visualizer(im[:, :, ::-1],
                  metadata=test_metadata,
                  scale=0.8
                  )
    out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
    cv2.imshow(out.get_image()[:, :, ::-1])
```

با استفاده از `ColorMode.IMAGE_BW` می توانیم رنگ ها را از اشیایی که شناسایی نمی شوند حذف کنیم و می توانیم نمونه ای از پیش بینی مدل را مشاهده کنیم.

```
[ ] from detectron2.utils.visualizer import ColorMode
import glob

for imageName in glob.glob('/content/Number_finder-5/train/*.jpg'):
    im = cv2.imread(imageName)
    outputs = predictor(im)
    v = Visualizer(im[:, :, ::-1],
                  metadata=test_metadata,
                  scale=0.8
                  )
    out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
    cv2.imshow(out.get_image()[:, :, ::-1])
```

خروجی:



دسترسی به گوگل درایو برای آپلود کردن تصویر تست:

```
[ ] from google.colab import drive
drive.mount('/content/gdrive')

Mounted at /content/gdrive

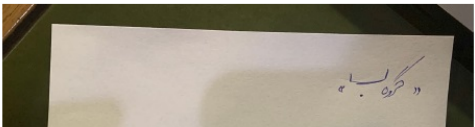
[ ] !ls gdrive

MyDrive

[ ] !ls MyDrive

ls: cannot access 'MyDrive': No such file or directory

im = cv2.imread("./test.jpg")
cv2.imshow(im)
```



نمایش نتیجه مدل روی تصویر تست: توسط اجرای predictor روی تصویر تست ریسایز شده (برای گرفتن نتیجه بهتر)

predictor = DefaultPredictor(cfg)

```
from matplotlib import pyplot as plt
import random
from detectron2.utils.visualizer import Visualizer
v = Visualizer(im[:, :, ::-1], MetadataCatalog.get(cfg.DATASETS.TRAIN[0]), scale=5)
out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
plt.imshow(out.get_image()[:, :, ::-1])
cv2.imwrite("final.jpg", out.get_image()[:, :, ::-1])
```


با توجه به اینکه معادلات با یک فلش شروع میشوند، پس میتوان مختصات فلش ها را به عنوان جایی در نظر گرفت که معادلات را از طریق آن ها پیدا کرد. به اندازه تعداد فلش های پیدا شده لیست خالی ایجاد میکنیم.

حال مختصات اولین استارت را در نظر میگیریم . مختصات y_1 آن را به عنوان کمترین میزان y و مختصات y_2 را به عنوان بیشترین میزان y در نظر میگیریم. با توجه به اینکه ممکن است به خاطر سایز نوشتار مربع ها خیلی کوچک شوند یک مقدار ثابت از y_{min} کم و به همان اندازه به y_{max} می افزاییم. این کار سبب میشود که محدوده گسترده تری برای جستجو مدنظر قرار گیرد. با توجه به اینکه مختصات ها بر اساس X مرتب شده اند پس تمامی کاراکتر هایی که در یک معادله قرار دارند بعد از آن قرار دارد. به همین دلیل مختصات مربع هایی را در نظر میگیریم که بعد از فلش باشند. پس از آن به دنبال مختصات مربع هایی میگردیم که که y مرکز آن ها بین y_{min} و y_{max} باشد. با پیدا شدن اولین مختصات دیگر مختصات فلش را به عنوان مرجع مد نظر قرار نمیدهیم و y های مربع انتخاب شده را به عنوان مرجع برای پیدا کردن مربع بعدی استفاده میشود. فلش فقط برای شناسایی معادلات استفاده شده و مختصات آن را ذخیره نمیکنیم. همین فرآیند گفته شده در بالا به ازای انتخاب مربع های بعدی نیز اتفاق میفتد تا جایی که با پیدا نشدن هیچ مربع جدیدی که در بازه قرارگیرد معادله فعلی تمام شده و مختصات فلش دیگری برای پیدا کردن مختصات بعدی استفاده شود.

```
[ ] classes=np.array(outputs["instances"].pred_classes.to("cpu"))
    boxes = outputs["instances"].get_fields().get("pred_boxes").to("cpu")

[ ] boxes_temp=boxes.tensor.numpy()
    box=[]
    for i in range(len(boxes)):
        box.append(np.append(boxes_temp[i],boxes[i].get_centers().numpy()))
    boxes = np.array(box)
    sorted_box = boxes[boxes[:,4].argsort()]

[ ] box=[]
    for i in range(len(sorted_box)):
        indices = np.where(boxes == sorted_box[i])[0][0]
        box.append((sorted_box[i],classes[indices]))
    box=np.array(box)
```

```
[ ] equation_num=0
equations=[]
for i in range(len(classes)):
    equations.append([])
for i in classes:
    y_min=5000
    y_max=0
    choosen_box = box[i]
    indice_k = i
    while True:
        found= False
        y_min = min(choosen_box[0][1]-40,y_min)
        y_max = max(choosen_box[0][3]+40,y_max)
        possible = box[indice_k+1:]
        for j in range(len(possible)):
            y=(possible[j][0][5])
            if y_min <= y <= y_max:
                if possible[j][1] == 5:
                    continue
                if choosen_box[1] != 5:
                    equations[equation_num].append(choosen_box)
                    indice_k = indice_k+j+1
                    choosen_box = box[indice_k]
                    found =True
                    break
        if not found:
            if choosen_box[1] != 5:
                equations[equation_num].append(choosen_box)
            equation_num+=1
            break
```

حال با پیدا شدن معادلات به سراغ یافتن جواب آن ها میرویم. با توجه به اینکه مختصات قبل از اضافه شدن به معادلات مرتب شده اند نیاز به ایجاد تغییر در آن ها نیست. از ابتدایی ترین مختصات در لیست معادله شروع میکنیم و کلاس پیدا شده به ازای آن مختصات را تفسیر کرده و مقدار آن را به صورت یک رشته کاراکتری (استرینگ) ذخیره میکنیم. در پایان یک رشته مانند "۶+۲" خواهیم داشت. حال با استفاده از eval زبان پایتون حاصل این رشته کاراکتری را محاسبه میکنیم. مختصات کمترین و بیشترین x , y در یک معادله را برای کشیدن مربع به دور آن ها ذخیره میکنیم.

```

answers=[]
for i in equations:
    temp = np.array(i)
    string=""
    min_y= float("inf")
    max_y=float("-inf")
    min_x= float("inf")
    max_x=float("-inf")
    for j in temp:
        if j[0][3]>max_y:
            max_y=j[0][3]
        if j[0][1]<min_y:
            min_y=j[0][1]
        if j[0][2]>max_x:
            max_x=j[0][2]
        if j[0][0]<min_x:
            min_x=j[0][0]
        if j[1]==1:
            string+="4"
        if j[1]==2:
            string+="-"
        if j[1]==3:
            string+="+"
        if j[1]==4:
            string+="6"
        if j[1]==5:
            continue
        if j[1]==6:
            string+="2"
    try:
        answers.append(np.array([eval(string),(min_x,min_y,max_x,max_y)]))
    except:
        continue

```

حال با داشتن مختصات مورد نیاز و حاصل نتیجه را بر روی تصویر ورودی درج میکنیم.

```
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from PIL import Image

im = Image.open('./test6.jpg')
im=im.resize((1000,1000))

# Create figure and axes
fig, ax = plt.subplots(figsize=(15, 15))
# Display the image
ax.imshow(im)
ax.axis('off')
# Coordinates of rectangle vertices
# in clockwise order
colors=["r","g","b","#7E14BE"]
for i in range(len(answers)):
    temp1=answers[i][1]
    xs = [temp1[0]-15,temp1[0]-15,temp1[2]+15,temp1[2]+15,temp1[0]-15]
    ys = [temp1[1]-10,temp1[3]+10,temp1[3]+10,temp1[1]-10,temp1[1]-10]
    ax.plot(xs, ys,colors[i])
    ax.text(temp1[2]+20,temp1[1]+20,str(answers[i][0]),color=colors[i],fontsize=15)
fig.show()
fig.savefig("result.jpg")
```


نتیجه تصویر تست داده شده در صورت پروژه

$$\Rightarrow 2 + 4 + 6^{12}$$
$$\Rightarrow 6 - 2 - 2 - 4 + 4^2$$
$$\Rightarrow 4 + 2^6$$
$$\Rightarrow 6 - 6 \times 6^6$$

به خوبی دیده میشود که توانسته ایم با استفاده از آجکت دیتکشن دیتکرون ۲ و تحلیل نتایج آن یک سیستم حل معادله بسازیم که با دقت خوبی بتواند کار کند.

لینک نوتبوک:

https://drive.google.com/file/d/view?usp=sharing&HqOI_cHI7ht3FVHcdv2-PsA0mrGGq12