Member-only story

COCO and Pascal VOC data format for Object detection

Understanding annotation data formats for computer vision

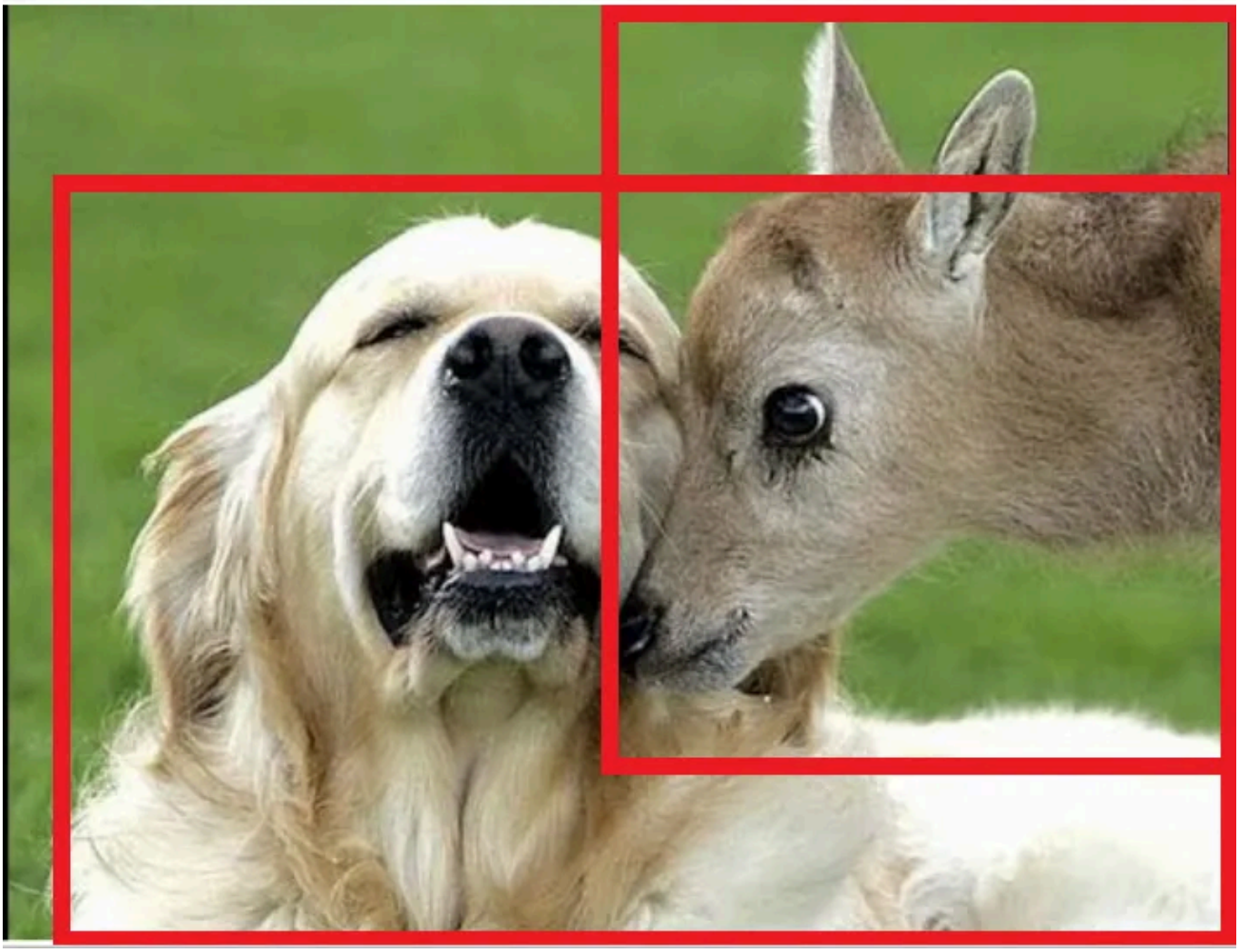
Renu Khandelwal · Follow

Published in Towards Data Science · 6 min read · Dec 7, 2019

In this article, we will understand two popular data formats: COCO data format and Pascal VOC data formats. These data formats are used for annotating objects found in a data set used for computer vision. we will especially focus on annotations for object detection

One of the most important tasks in computer vision is to label the data. There are several tools available where you can load the images, label the objects using per-instance segmentation. This aids in precise object localization using bounding boxes or masking using polygons. This information is stored in annotation files.

Annotation file/files can be in COCO or Pascal VOC data formats.



What is COCO?

COCO is large scale images with Common Objects in Context (COCO) for object detection, segmentation, and captioning data set. COCO has 1.5 million object instances for 80 object categories

COCO has 5 annotation types used for

- object detection
- keypoint detection
- stuff segmentation
- panoptic segmentation

- image captioning

COCO stores annotations in a JSON file. Let's look at the JSON format for storing the annotation details for the bounding box. This will help to create your own data set using the COCO format.

The basic building blocks for the JSON annotation file is

- **info:** contains high-level information about the dataset.
- **licenses:** contains a list of image licenses that apply to images in the dataset.
- **categories:** contains a list of categories. Categories can belong to a supercategory
- **images:** contains all the image information in the dataset without bounding box or segmentation information. image ids need to be unique
- **annotations:** list of every individual object annotation from every image in the dataset

```
{
  "info": info,
  "licenses": [license],
  "categories": [category],
  "images": [image],
  "annotations": [annotation]
}
```

Sample COCO JSON format

We can create a separate JSON file for train, test and validation dataset.

let's dive into each section

Info:

Provides information about the dataset.

```
"info"
{
"year": int,
"version": str,
"description": str,
"contributor": str,
"url": str,
"date_created": datetime,
}
```

```
"info":
    {
        "year": 2019,
        "version": "1.0",
        "description": "Flower and Fruits dataset",
        "contributor": "Flowers Inc.",
        "url": "http://test.org",
        "date_created": "2019/12/04"
    }
```

template and example for info section of the JSON for COCO

Licenses:

We can provide a list of different image licenses used in the dataset.

```
licenses
[{
"id": int,
"name": str,
"url": str,
}]
```

```
"licenses":
[{
        "id": 1,
        "name": "Free License",
        "url": "http://flower.org/",
}]
```

template and example for Licenses section of the JSON for COCO

Categories:

Each category id must be unique. A category can belong to a super-category. As an example, if we have data set to identify flowers and fruits. Flower will be super-category and rose, lily, tulip would be the name of the flowers we want to detect.

```
Categories
[{
"id": int,
 "name": str,
 "supercategory": str,
}]
```

```
"categories":
[
    {"id": 1,"name": "rose", "supercategory": "flower"},
    {"id": 2,"name": "tulip", "supercategory": "flower"},
    {"id": 10,"name": "Apple", "supercategory": "fruit"}
]
```

template and example for Categories section of the JSON for COCO

Images:

Contains list of all the images in the dataset. Image id should be unique. flickr_url, coco_url and date_captured are optional

```
image{
"id": int,
 "width": int,
 "height": int,
 "file_name": str,
 "license": int,
 "flickr_url": str,
 "coco_url": str,
 "date_captured": datetime,
}
```

```
"images": [
    {
        "id": 397133,
        "width": 640,
        "height": 640,
        "file_name": "101.jpg",
        "license":1,
         "date_captured": "2019-12-04 17:02:52"
    },
    {
         "id": 397122,
        "height": 640,
        "width": 640,
        "file_name": "102.jpg",
        "license": 1,
        "date_captured": "2019-12-04 17:02:52"
    }
    ]
```

template and example for images section of the json for COCO

Annotations:

Contains list of each individual object annotation from every single image in the dataset. This is the section that contains the bounding box output or object segmentation for object detection

If an image has 4 objects that we want to detect then we will have annotations for all 4 objects.

If the entire dataset consists of 150 images and has a total of 200 objects then we will have 200 annotations.

segmentation contains the x and y coordinates for the vertices of the polygon around every object instance for the segmentation masks.

area is the area of the bounding box. It is a pixel value

iscrowd: If we have a single object segmentation then iscrowd is set to zero. For a collection of objects present in the image, we set iscrowd=1, in which case RLE is used.

RLE is Run Length Encoding. When iscrowd=1, then we add attribute **counts** and **size** in the segmentation section. This is the second segmentation in the example below

A single object (iscrowd=0) if occluded may require multiple polygons.

imageid: It is the id of the image which contains the objects for which we are specifying the annotations . The imageid corresponds to the imageid that we have in the image section

bbox : Bounding box in COCO is the x and y co-ordinate of the top left and the height and width. Pascal VOC bounding box is the x and y co-ordinates of the top left and x and y co-ordinates of the bottom right edge of the rectangle.

COCO Bounding box: (*x-top left, y-top left, width, height*)

Pascal VOC Bounding box :(*x-top left, y-top left,x-bottom right, y-bottom right*)

category: It is the category of the object that we have earlier specified in the categories section

id: It is the unique id for the annotations

```
annotation{
"id": int,
"image_id": int,
"category_id": int,
"segmentation": RLE or [polygon],
"area": float,
"bbox": [x,y,width,height],
"iscrowd": 0 or 1,
}
```

```
"annotations": [
    {
        "segmentation":
[[510.66,423.01,511.72,420.03,...,510.45,423.01]],
        "area": 702.10,
        "iscrowd": 0,
        "image_id":  397133,
        "bbox": [433.07,355.93,138.65,228.67],
        "category_id": 18,
        "id": 1768
    },
    {
        "segmentation":
      {
        "counts":[12,56,198,10]
        "size":[120, 240]
      }
        "area": 500.2,
        "iscrowd": 1,
        "image_id": 397122,
        "bbox": [473.07,395.93,38.65,28.67],
        "category_id": 18,
        "id": 1768
    }]
```

template and example for annotations section of the JSON for COCO

What is Run Length Encoding(RLE)?

RLE is a compression method that works by replacing repeating values by the number of times they repeat.

For example 0 11 0111 00 would become 1 2 1 3 2.

COCO data format provides segmentation masks for every object instance as shown above in the segmentation section. This creates efficiency issues to

- store the masks compactly and
- to perform mask computations efficiently.

We use using a Run Length Encoding (RLE) scheme to address both the issues.

The size of the RLE representation is proportional to the number of boundary pixels of a mask. Operations such as area, union, or intersection will be computed efficiently on the RLE.

Pascal Visual Object Classes(VOC)

Pascal VOC provides standardized image data sets for object detection

Difference between COCO and Pacal VOC data formats will quickly help understand the two data formats

- Pascal VOC is an XML file, unlike COCO which has a JSON file.
- In Pascal VOC we create a file for each of the image in the dataset. In COCO we have one file each, for entire dataset for training, testing and validation.
- The bounding Box in Pascal VOC and COCO data formats are different

COCO Bounding box: (*x-top left, y-top left, width, height*)

Pascal VOC Bounding box :(*xmin-top left, ymin-top left,xmax-bottom right, ymax-bottom right*)

```
<annotation>
        <folder>Kangaroo</folder>
        <filename>00001.jpg</filename>
        <path>./Kangaroo/stock-12.jpg</path>
        <source>
                <database>Kangaroo</database>
        </source>
        <size>
                <width>450</width>
                <height>319</height>
                <depth>3</depth>
        </size>
        <segmented>0</segmented>
        <object>
                <name>kangaroo</name>
                <pose>Unspecified</pose>
                <truncated>0</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>233</xmin>
                        <ymin>89</ymin>
                        <xmax>386</xmax>
                        <ymax>262</ymax>
                </bndbox>
        </object>
</annotation>
```

Sample Pascal VOC

Some of the key tags for Pascal VOC are explained below

Folder:

Folder that contains the images

Filename:

Name of the physical file that exists in the folder

Size:

Contain the size of the image in terms of width, height and depth. If the image is black and white then the depth will be 1. For color images, depth will be 3

Object:

Contains the object details. If you have multiple annotations then the object tag with its contents is repeated. The components of the object tags are

- name
- pose
- truncated
- difficult
- bndbox

name:

This is the name of the object that we are trying to identify

truncated:

Indicates that the bounding box specified for the object does not correspond to the full extent of the object. For example, if an object is visible partially in the image then we set truncated to 1. If the object is fully visible then set truncated to 0

difficult:

An object is marked as difficult when the object is considered difficult to recognize. If the object is difficult to recognize then we set difficult to 1 else set it to 0

bounding box:

Axis-aligned rectangle specifying the extent of the object visible in the image.

This article should help understand the details of the two popular data formats used in computer vision

References:

http://cocodataset.org/#download

COCO - Common Objects in Context

Edit description

cocodataset.org

how object segmentation are written in dataset json files? · Issue #111 · cocodataset/cocoapi

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or...

github.com

https://pjreddie.com/media/files/VOC2012_doc.pdf

https://arxiv.org/pdf/1405.0312.pdf

Computer Vision | Object Detection | Coco | Annotations | Bounding Box

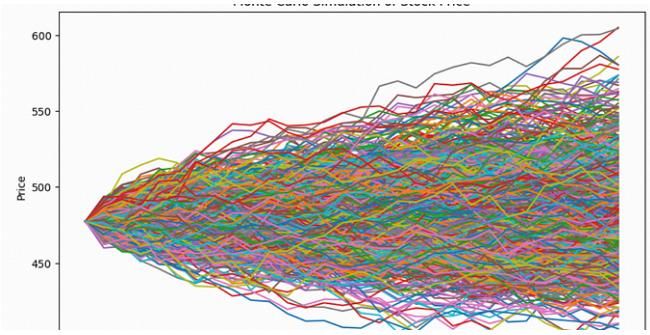
Written by Renu Khandelwal

Follow

6.8K Followers · Writer for Towards Data Science

A Technology Enthusiast who constantly seeks out new challenges by exploring cutting-edge technologies to make the world a better place!

More from Renu Khandelwal and Towards Data Science



Renu Khandelwal

Intuitive Explanation of Monte Carlo Simulation

Simple Explanation of Monte Carlo Simulation and Implementation using Python

Tim Sumner in Towards Data Science

A New Coefficient of Correlation

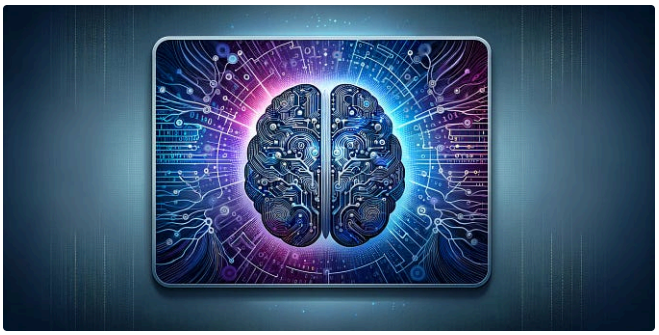
What if you were told there exists a new way to measure the relationship between two...

7 min read · Apr 18, 2024

81 1

10 min read · Mar 31, 2024

3K 36



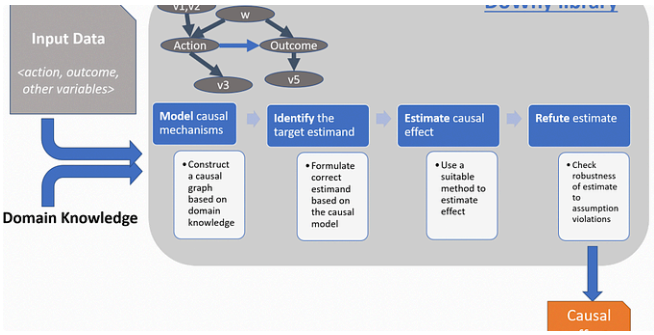
Cristian Leo in Towards Data Science

The Math Behind Neural Networks

Dive into Neural Networks, the backbone of modern AI, understand its mathematics,...

28 min read · Mar 29, 2024

3K 20



Renu Khandelwal

Exploring Causality with DoWhy

Perform Causal Inference Using Python Step by Step with DoWhy

10 min read · Apr 15, 2024

18

See all from Renu Khandelwal

See all from Towards Data Science

Recommended from Medium

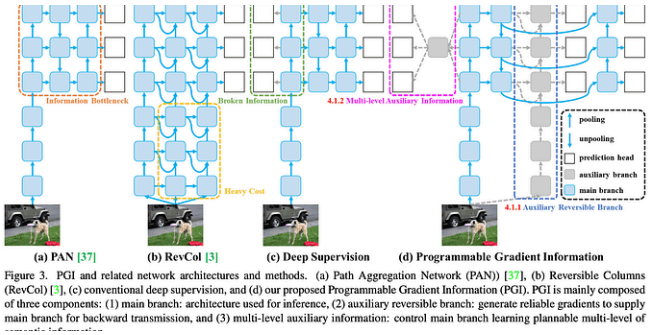


Figure 3. PGI and related network architectures and methods. (a) Path Aggregation Network (PAN)) [37], (b) Reversible Columns (RevCol) [3], (c) conventional deep supervision, and (d) our proposed Programmable Gradient Information (PGI). PGI is mainly composed of three components: (1) main branch: architecture used for inference, (2) auxiliary reversible branch: generate reliable gradients to supply main branch for backward transmission, and (3) multi-level auxiliary information: control main branch learning plannable multi-level of

Rizwan Ahmed

Train YOLOv9 on custom dataset

Are you eager to dive into the world of object detection and train your own custom models...

5 min read · Mar 9, 2024

Betül Samancı

Object Detection with Yolov8

In the world of image processing, the ability of computers to detect and localize objects in...

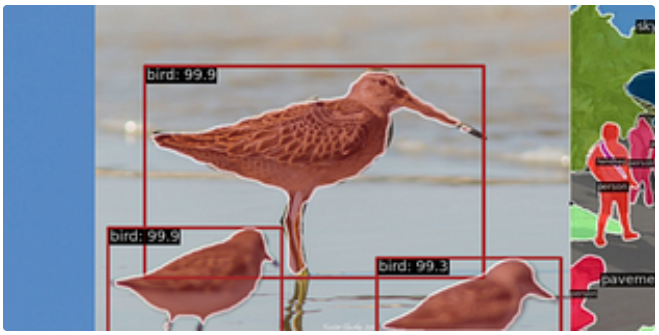
7 min read · Mar 12, 2024

5

Lists

Natural Language Processing

1402 stories · 899 saves



George Pearse

How to train a model with MMDetection

Raj Pulapakura

Image Segmentation—A Beginner's Guide

MMDetection is an excellent tool, I've used Detectron2 and Pytorch-Lightning with the...

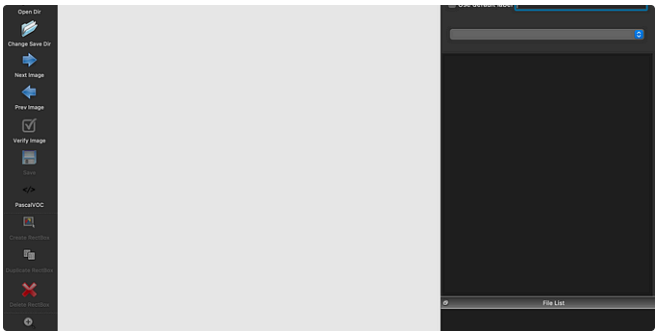
4 min read · Dec 16, 2023

9

The essentials of Image Segmentation + implementation in TensorFlow

6 min read · Feb 4, 2024

58



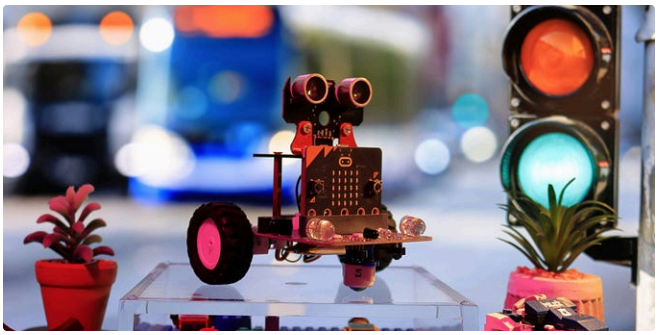
Samin karki

Installing and using Labelimg to annotate images

Ever since I heard about the YOLO object detection, I always wanted to create a mobil...

3 min read · Nov 3, 2023

1 1



Oliver Lövström

Fine-tuning YOLOv8

Step-by-step guide for fine-tuning YOLOv8 using your own datasets in Google Colab

· 3 min read · Feb 8, 2024

334 2

See more recommendations