IMMERSIVE LIMIT

Create COCO Annotations From Scratch

WANT TO CREATE A CUSTOM DATASET?

👉Check out the Courses page for a complete, end to end course on creating a COCO dataset from scratch.

TABLE OF CONTENTS

This page has grown to be really long…

👇Jump to whichever section seems most interesting:

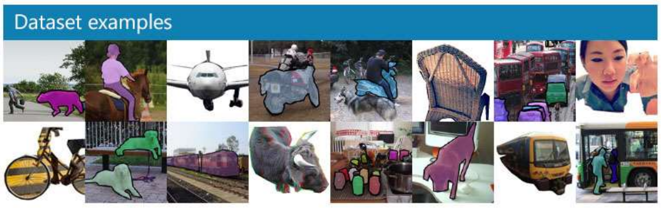
IMMERSIVE LIMIT

WHAT IS THE COCO DATASET?

COCO annotations are inspired by the Common Objects in Context (COCO) dataset. According to cocodataset.org/#home:

> "COCO is a large-scale object detection, segmentation, and captioning dataset. COCO has several features: Object segmentation, Recognition in context, Superpixel stuff segmentation, 330K images (>200K labeled), 1.5 million object instances, 80 object categories, 91 stuff categories, 5 captions per image, 250,000 people with keypoints."

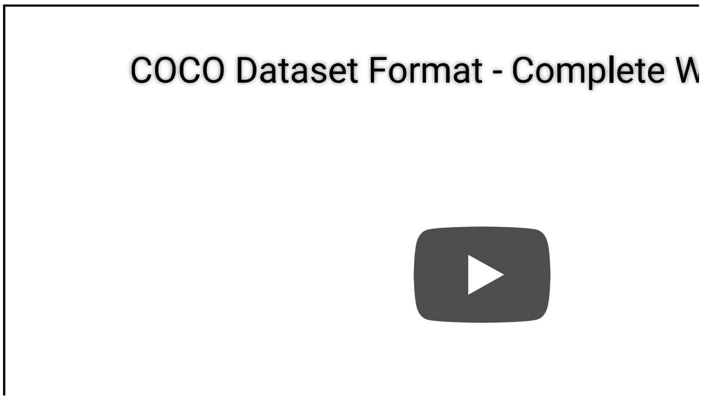
It is one of the best image datasets available, so it is widely used in cutting edge image recognition artificial intelligence research. It is used in open source projects such as Facebook Research's Detectron, Matterport's Mask R-CNN, endernewton's Tensorflow Faster RCNN for Object Detection, and others.



A few example images from http://cocodataset.org/#home

Because it is used by so many projects, you probably want to know how to make your own, so let's quit wasting time.

IMMERSIVE LIMIT



A detailed walkthrough of the COCO Dataset JSON Format, specifically for object detection (instance segmentations). The first step toward making your own COCO dataset is understanding how it works. This video should help.

If you're not a video person or want more detail, keep reading.

The COCO dataset is formatted in JSON and is a collection of "info", "licenses", "images", "annotations", "categories" (in most cases), and "segment info" (in one case).

```
{
    "info": {...},
    "licenses": [...],
    "images": [...],
    "annotations": [...],
    "categories": [...], <-- Not in Capt
    "segment_info": [...] <-- Only in Pa
}
```

INFO

The "info" section contains high level information about the dataset. If you are creating your own dataset, you can fill in whatever is appropriate.

IMMERSIVE LIMIT

```
    "description": "COCO 2017 Dataset",
    "url": "http://cocodataset.org",
    "version": "1.0",
    "year": 2017,
    "contributor": "COCO Consortium",
    "date_created": "2017/09/01"
}
```

LICENSES

The “licenses” section contains a list of image licenses that apply to images in the dataset. If you are sharing or selling your dataset, you should make sure your licenses are correctly specified and that you are not infringing on copyright.

```
"licenses": [
    {
        "url": "http://creativecommons.o
        "id": 1,
        "name": "Attribution-NonCommerci
    },
    {
        "url": "http://creativecommons.o
        "id": 2,
        "name": "Attribution-NonCommerci
    },
    ...
]
```

IMAGES

The “images” section contains the complete list of images in your dataset. There are no labels, bounding boxes, or segmentations specified in this part, it's simply a list of images and information

IMMERSIVE LIMIT

learning application probably will only need the file_name.

Note that image ids need to be unique (among other images), but they do not necessarily need to match the file name (unless the deep learning code you are using makes an assumption that they'll be the same... developers are lazy, it wouldn't surprise me).

```
"images": [
    {
        "license": 4,
        "file_name": "000000397133.jpg",
        "coco_url": "http://images.cocod
        "height": 427,
        "width": 640,
        "date_captured": "2013-11-14 17:
        "flickr_url": "http://farm7.stat
        "id": 397133
    },
    {
        "license": 1,
        "file_name": "000000037777.jpg",
        "coco_url": "http://images.cocod
        "height": 230,
        "width": 352,
        "date_captured": "2013-11-14 20:
        "flickr_url": "http://farm9.stat
        "id": 37777
    },
    ...
]
```

FIVE COCO ANNOTATION TYPES

> COCO has five annotation types: for object detection, keypoint detection, stuff segmentation, panoptic segmentation, and image captioning. The annotations are stored using JSON.

The documentation on the COCO annotation format isn't crystal clear, so I'll break them down as simply as I can. Each one is a little different.

OBJECT DETECTION (SEGMENTATION)

http://cocodataset.org/#detection-2018

This is the most popular one; it draws shapes around objects in an image. It has a list of categories and annotations.

CATEGORIES

The "categories" object contains a list of categories (e.g. dog, boat) and each of those belongs to a supercategory (e.g. animal, vehicle). The original COCO dataset contains 90 categories. You can use the existing COCO categories or create an entirely new list of your own. Each category id must be unique (among the rest of the categories).

```
"categories": [
    {"supercategory": "person","id": 1,"
    {"supercategory": "vehicle","id": 2,
    {"supercategory": "vehicle","id": 3,
    {"supercategory": "vehicle","id": 4,
```

IMMERSIVE LIMIT

```
    {"supercategory": "indoor","id": 89,
    {"supercategory": "indoor","id": 90,
]
```

ANNOTATIONS

The “annotations” section is the trickiest to understand. It contains a list of every individual object annotation from every image in the dataset. For example, if there are 64 bicycles spread out across 100 images, there will be 64 bicycle annotations (along with a ton of annotations for other object categories). Often there will be multiple instances of an object in an image. Usually this results in a new annotation item for each one.

I say “usually” because regions of interest indicated by these annotations are specified by “segmentations”, which are usually a list of polygon vertices around the object, but can also be a run-length-encoded (RLE) bit mask. Typically, RLE is used for groups of objects (like a large stack of books). I’ll explain how this works later in the article.

Area is measured in pixels (e.g. a 10px by 20px box would have an area of 200).

Is Crowd specifies whether the segmentation is for a single object or for a group/cluster of objects.

The image id corresponds to a specific image in the dataset.

The COCO bounding box format is [top left x position, top left y position, width, height].

The category id corresponds to a single category specified in the categories section.

IMMERSIVE LIMIT

The following JSON shows 2 different annotations.

1. The first annotation:
    - Has a segmentation list of vertices (x, y pixel positions)
    - Has an area of 702 pixels (pretty small) and a bounding box of [473.07,395.93,38.65,28.67]
    - Is not a crowd (meaning it's a single object)
    - Is category id of 18 (which is a dog)
    - Corresponds with an image with id 289343 (which is a person on a strange bicycle and a tiny dog)
2. The second annotation:
    - Has a Run-Length-Encoding style segmentation
    - Has an area of 220834 pixels (much larger) and a bounding box of [0,34,639,388]
    - Is a crowd (meaning it's a group of objects)
    - Is a category id of 1 (which is a person)
    - Corresponds with an image with id 250282 (which is a vintage class photo of about 50 school children)

```
"annotations": [
    {
```

IMMERSIVE LIMIT

```
            "iscrowd": 0,
            "image_id": 289343,
            "bbox": [473.07,395.93,38.65,28.(
            "category_id": 18,
            "id": 1768
        },
        ...
        {
            "segmentation": {
                "counts": [179,27,392,41,…,5!
                "size": [426,640]
            },
            "area": 220834,
            "iscrowd": 1,
            "image_id": 250282,
            "bbox": [0,34,639,388],
            "category_id": 1,
            "id": 900100250282
        }
]
```

KEYPOINT DETECTION FORMAT

http://cocodataset.org/#keypoints-2018

Keypoints add additional information about a segmented object. They specify a list of points of interest, connections between those points, where

IMMERSIVE LIMIT

CATEGORIES

As of the 2017 version of the dataset, there is only one category ("person") in the COCO dataset with keypoints, but this could theoretically be expanded to any category that might have different points of interest. For example a shark (tail, fins, eyes, gills, etc) or a robotic arm (grabber, joints, base).

In the case of a person, "keypoints" indicate different body parts. The "skeleton" indicates connections between points. For example, [16, 14] means "left_ankle" connects to "left_knee".

```
"categories": [
    {
        "supercategory": "person",
        "id": 1,
        "name": "person",
        "keypoints": [
            "nose","left_eye","right_eye
            "left_shoulder","right_shoul
            "left_wrist","right_wrist","
            "left_knee","right_knee","le
        ],
        "skeleton": [
            [16,14],[14,12],[17,15],[15,
            [6,8],[7,9],[8,10],[9,11],[2
        ]
    }
]
```

ANNOTATIONS

Annotations for keypoints are just like in Object x and y indicate pixel positions in the image.

v indicates visibility— v=0: not labeled (in which case x=y=0), v=1: labeled but not visible, and v=2: labeled and visible

229, 256, 2 means there's a keypoint at pixel x=229, y=256 and 2 indicates that it is a visible keypoint

```
"annotations": [
    {
        "segmentation": [[204.01,306.23,
        "num_keypoints": 15,
        "area": 5463.6864,
        "iscrowd": 0,
        "keypoints": [229,256,2,...,223,
        "image_id": 289343,
        "bbox": [204.01,235.08,60.84,177
        "category_id": 1,
        "id": 201376
    }
]
```

STUFF SEGMENTATION FORMAT

http://cocodataset.org/#stuff-2018

Stuff segmentation is identical to object detection (except is_crowd is unnecessary). You can learn more about it here: http://cocodataset.org/#stuff-eval

PANOPTIC SEGMENTATION

http://cocodataset.org/#panoptic-2018

I'm still working on this part (as of Jan 19, 2019). Check back soon!

IMAGE CAPTIONING

Image caption annotations are pretty simple. There are no categories in this JSON file, just annotations with caption descriptions. Both of the pictures I checked actually had 4 separate captions for each image, presumably from different people.

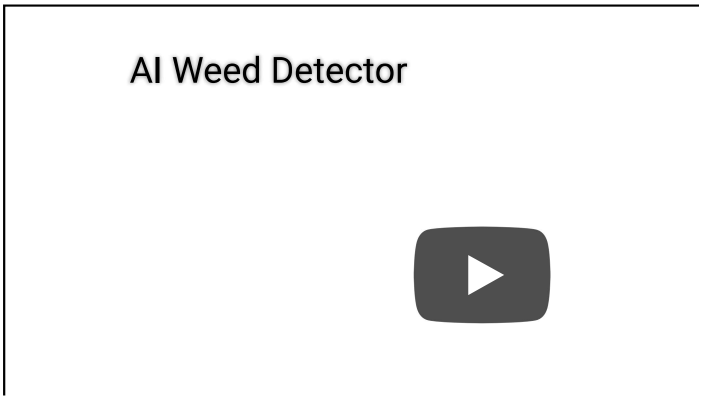
```
"annotations": [
    {
        "image_id": 289343,
        "id": 433580,
        "caption": "A person riding a ve
    },
    ...
    {
        "image_id": 250282,
        "id": 511309,
        "caption": "A group of school ch
    },
]
```

CREATING A CUSTOM COCO DATASET

WITH SYNTHETIC GENERATION

If you want to save 100 to 1000 hours per project, I recommend you create your COCO dataset synthetically. I even created a course to teach you how. Essentially, you write code that composes foreground images of objects over top of random image backgrounds. It may not work for every application, but you might be surprised at what can be achieved.

Here's an example of a synthetic COCO dataset I created to detect lawn weeds:



And here's another example, where I made a custom COCO dataset of cigarette butts and was able to detect them in images:

[Using Mask R-CNN with a Custom COCO-like Dataset]

Check out this tutorial to learn how to use Mask R-CNN on COCO-like datasets.

Complete Guide to CreatingCOCO Datasets

Python.

MANUALLY, USING VERTICES

You can also, of course, create annotations with vertices. This is how 99%+ of the original COCO dataset was created.

The original researchers used Amazon Mechanical Turk to hire people for SUPER cheap to use a web app and tediously draw shapes around objects. The code they used can be found here https://github.com/tylin/coco-ui, but I found it to be fairly unusable since it's all hooked up to AWS and Mechanical Turk. You might be able to tweak it for your own purposes if you really want to.

There's another, more recent, open-source project: COCO Annotator that is worth looking into instead. I've used it briefly and it seems very good. https://github.com/jsbroks/coco-annotator

WHY YOU SHOULDN'T TRY TO CREATE YOUR OWN THIS WAY

I haven't taken the time to find a great solution here because I'm more interested in creating synthetic datasets. Why? Because if it takes me 2 minutes on average to manually annotate an image and I have to annotate at least 2000 labeled images for a *small* dataset (COCO has 200K labeled images), it would take me 4000 minutes, which is over 66 straight hours. I'll pass.

USING BIT MASKS

IMMERSIVE LIMIT

generate polygons if you start with a masked image. This is particularly interesting if you have a synthetic dataset (e.g. created by a game engine) that outputs masks.

I'm going to use the following two images for an example. The COCO dataset only contains 90 categories, and surprisingly "lamp" is not one of them. I'm going to create this COCO-like dataset with 4 categories: houseplant, book, bottle, and lamp. (The first 3 are in COCO)

The first step is to create bit masks for each item of interest in the scene. That's 5 objects between the 2 images here. In the method I'm teaching here, it doesn't matter what color you use, as long as there is a distinct color for each object. You also don't need to be consistent with what colors you use, but you will need to make a note of what colors you used for each category. I used GIMP to make these images.

Important note #1: Even if there were 2 books in one scene, they would need different colors, because overlapping books would be impossible to distinguish by the code we are going to write.

Important note #2: Make sure each pixel is a solid color. Some image applications will perform edges below.

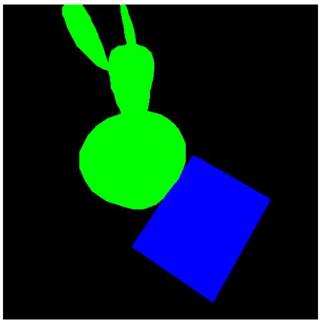
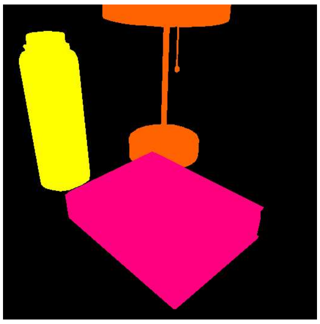



Image 1 colors: (0, 255, 0): houseplant; (0, 0, 255): book
Image 2 colors: (255, 255, 0): bottle; (255, 0, 128): book; (255, 100, 0): lamp

I've also assigned the following categories (you can choose any integers you want): 1 = houseplant, 2 = book, 3 = bottle, 4 = lamp

HIGH LEVEL OVERVIEW OF CREATING ANNOTATIONS

This is the only difficult part, really. Here are the steps required for this method:

1. Write code to automatically split up the image into individual masks
2. Write code to create polygons out of each individual mask
3. Convert the information to JSON

Create sub-masks

Here's a python function that will take in a mask Image object and return a dictionary of sub-masks, keyed by RGB color. Note that it adds a padding pixel which we'll account for later.

```python
def create_sub_masks(mask_image):
    width, height = mask_image.size

    # Initialize a dictionary of sub-mas
    sub_masks = {}
    for x in range(width):
        for y in range(height):
            # Get the RGB values of the
            pixel = mask_image.getpixel(

            # If the pixel is not black.
            if pixel != (0, 0, 0):
                # Check to see if we've
                pixel_str = str(pixel)
                sub_mask = sub_masks.get
                if sub_mask is None:
                   # Create a sub-mask (
                    # Note: we add 1 pix
                    # because the contou
                    # where pixels bleed
                    sub_masks[pixel_str]

                # Set the pixel value to
                sub_masks[pixel_str].put

    return sub_masks
```

Create sub-mask annotation

Here's a python function that will take a sub-mask, create polygons out of the shapes inside, and then return an annotation dictionary. This is where we remove the padding mentioned above.

IMMERSIVE LIMIT

```
from skimage import measure
from shapely.geometry import Polygon, Mu

def create_sub_mask_annotation(sub_mask,
    # Find contours (boundary lines) aro
    # Note: there could be multiple conte
    # is partially occluded. (E.g. an ele
    contours = measure.find_contours(sub_

    segmentations = []
    polygons = []
    for contour in contours:
        # Flip from (row, col) represent
        # and subtract the padding pixel
        for i in range(len(contour)):
            row, col = contour[i]
            contour[i] = (col - 1, row -

        # Make a polygon and simplify it
        poly = Polygon(contour)
        poly = poly.simplify(1.0, preser
        polygons.append(poly)
        segmentation = np.array(poly.exte
        segmentations.append(segmentatio

    # Combine the polygons to calculate
    multi_poly = MultiPolygon(polygons)
    x, y, max_x, max_y = multi_poly.boun
    width = max_x - x
    height = max_y - y
    bbox = (x, y, width, height)
    area = multi_poly.area

    annotation = {
        'segmentation': segmentations,
        'iscrowd': is_crowd,
```

IMMERSIVE LIMIT

```
        'id': annotation_id,
        'bbox': bbox,
        'area': area
    }

    return annotation
```

Putting it all together

Finally, we'll use these two functions on our images

```
import json

plant_book_mask_image = Image.open('/patl
bottle_book_mask_image = Image.open('/pa

mask_images = [plant_book_mask_image, bo

# Define which colors match which catego
houseplant_id, book_id, bottle_id, lamp_
category_ids = {
    1: {
        '(0, 255, 0)': houseplant_id,
        '(0, 0, 255)': book_id,
    },
    2: {
        '(255, 255, 0)': bottle_id,
        '(255, 0, 128)': book_id,
        '(255, 100, 0)': lamp_id,
    }
}

is_crowd = 0

# These ids will be automatically increa
```

```
# Create the annotations
annotations = []
for mask_image in mask_images:
    sub_masks = create_sub_masks(mask_ima
    for color, sub_mask in sub_masks.iter
        category_id = category_ids[image_
        annotation = create_sub_mask_anno
        annotations.append(annotation)
        annotation_id += 1
    image_id += 1

print(json.dumps(annotations))
```

Output JSON

And here's our output! (I omitted a lot of the vertices for readability)


```
[{
    "segmentation": [[228.0, 332.5, 248.
    "iscrowd": 0,
    "image_id": 1,
    "category_id": 1,
    "id": 1,
    "bbox": [95.5, -0.5, 201.0, 333.0],
    "area": 33317.25
},
{
    "segmentation": [[340.0, 483.5, 343.
    "iscrowd": 0,
    "image_id": 1,
    "category_id": 2,
    "id": 2,
    "bbox": [209.5, 244.5, 225.0, 239.0],
```

IMMERSIVE LIMIT

```
{
    "segmentation": [[95.0, 303.5, 108.0
    "iscrowd": 0,
    "image_id": 2,
    "category_id": 3,
    "id": 3,
    "bbox": [25.5, 42.5, 116.0, 261.0],
    "area": 22243.5
},
{
    "segmentation": [[282.0, 494.5, 415.
    "iscrowd": 0,
    "image_id": 2,
    "category_id": 2,
    "id": 4,
    "bbox": [100.5, 238.5, 324.0, 256.0]
    "area": 47241.0
},
{
    "segmentation": [[298.0, 263.5, 311.
    "iscrowd": 0,
    "image_id": 2,
    "category_id": 4,
    "id": 5,
    "bbox": [161.5, -0.5, 210.0, 264.0],
    "area": 14593.25
}]
```

PREVIEWING COCO ANNOTATIONS FOR AN IMAGE

time I went digging, so I decided to write my own. After a couple dead-end attempts to make a GUI application in Python for this, I ended up making something in Jupyter Notebook that lets you import a dataset as json and view segmentations in images.

To keep things simple, I created a subset of the COCO instances val2017 dataset that contains only 2 images.

Here's what these images will look like if you run your own Jupyter Notebook (Gist can't show the polygon segmentations for some reason).

IMMERSIVE LIMIT

COCO Image Viewer

This notebook will allow you to view details abou dataset and preview segmentations on annotated ima more about it at: http://cocodataset.org/ (http://cocodata

Note: Gist probably won't show the segmentations, bu this code in your own Jupyter Notebook, you'll see ther

The rest of the tutorial can be f http://www.immersivelimit.com/tutorials/create-coco-an from-scratch (http://www.immersivelimit.com/tutorials/c annotations-from-scratch)

In [1]:

```
import IPython
import os
import json
import random
import numpy as np
import requests
from io import BytesIO
import base64
from math import trunc
```

COCO_Image_Viewer.ipynb hosted with ♡ by GitHub view raw

Was this page helpful?

- Yes, I learned exactly what I needed.
- Needs improvement.

Your Email Address:

(If you want a response)

Feedback:

Add your feedback here.

Submit

PREVIOUS

Unity ML Agents | Visual Chameleons

NEXT

Unity ML-Agents Tutorial | AI Truffle Pigs! 🐷

ABOUT CONNECT

Powered by Squarespace

Privacy & Legal Policies