Guide to Creating KITTI Datasets

Author: Matthew Howlett
Document most recent update: September 24, 2019

1. Overview

This document is intended to guide a user in creating a complete KITTI dataset from scratch. This is done by first creating a COCO dataset (simpler to make), then using a python script to convert the COCO dataset into a KITTI dataset. The example dataset created in this tutorial is primarily made up of stop signs, but more signs are continually being added. In completing this tutorial, you will have the tools to create a custom KITTI dataset ready to be used for training a deep learning model.

2. Understanding the KITTI Format

Data Format Description
==========
Source/more info:
https://github.com/NVIDIA/DIGITS/blob/v4.0.0-rc.3/digits/extensions/data/objectDetection/README.md
==========

#Values	Name	Description
1	type	Describes the type of object: 'Car', 'Van', 'Truck', 'Pedestrian', 'Person_sitting', 'Cyclist', 'Tram', 'Misc' or 'DontCare'
1	truncated	Float from 0 (non-truncated) to 1 (truncated), where truncated refers to the object leaving image boundaries
1	occluded	Integer (0,1,2,3) indicating occlusion state: 0 = fully visible, 1 = partly occluded 2 = largely occluded, 3 = unknown
1	alpha	Observation angle of object, ranging [-pi..pi]
4	bbox	2D bounding box of object in the image (0-based index): contains left, top, right, bottom pixel coordinates
3	dimensions	3D object dimensions: height, width, length (in meters)
3	location	3D object location x,y,z in camera coordinates (in meters)
1	rotation_y	Rotation ry around Y-axis in camera coordinates [-pi..pi]
1	score	Only for results: Float, indicating confidence in detection, needed for p/r curves, higher is better.

Example KITTI label text file output:

stop_sign 0 0 0 803.5 77.5 911.5 189.5 0 0 0 0 0 0 0

3. Required resources

➢ Everything below

https://github.com/howl0893/custom-object-detection-datasets

Navigate to the link above to find all the resources mentioned below, organized into one master GitHub specifically made for this tutorial.

➢ CocoSynth Git Repo

https://github.com/akTwelve/cocosynth

The CocoSynth git repo contains the code required to create COCO dataset specific json files and image masks. You can clone it using git, or just download the zip file from Github.

➢ Anaconda

https://www.anaconda.com/distribution/

Anaconda is used for ease of setup of our Python environment with conda. It can be done using the windows command line, but the conda commands are provided in this tutorial so it suggested to use Anaconda. Setup your Python environment with conda using the following commands (after cloning/downloading the CocoSynth Repo):

```
conda create -n cocosynth python=3.6
conda activate cocosynth
conda install -c conda-forge shapely
pip install -r requirements.txt
```

Note: Perfectly okay to use Python 3.7 here. If you are setting up with the windows command line, use pip instead of conda. If you run into any issues while setting up copy the error code and description and paste into google. StackOverflow is typically a great site for debugging setup issues.

➢ Sublime Text (Optional)

https://www.sublimetext.com/

This is my prefered text editor for development. Feel free to use the code editor that you are most comfortable with.

➢ GIMP

https://www.gimp.org/downloads/

GIMP is an image editing software that will be used to annotate the images of your dataset. Download and install the most recent version.

➢ COCO to KITTI Conversion Script

https://github.com/dusty-nv/jetson-inference/blob/master/tools/coco2kitti.py
After creating the COCO dataset we will use this script to convert it into the KITTI format. Be sure to have this script downloaded and ready for when the time comes!

- ➢ Resize and Pad Image Script

https://www.coria.com/insights/blog/computer-vision/preparing-data-for-custom-object-detection-using-nvidia-digits
This script was found at the above directory, which is another good tutorial about preparing data for custom object detection using Nvidia Digits. If you scroll down to the section “Resizing images to adequate size” there is some python code that I just copied and pasted into a text editor, then saved as a python script.

4. Resize and Pad Collected Images

For the purpose of this tutorial images of stops signs were just collected over the internet using google images. Collect images in any way you see fit and compile them all into one folder. Resize and pad all of the images by running the size_pad_images.py python script located in the python folder of the creating-KITTI-datasets-master GitHub repo or found in the last bullet point of the “Required Resources” section of this document.

Example command:

```
python size_pad_images.py width height input_dir output_dir

python size_pad_images.py 1280 720
C:/users/matt/../datasets/stop_signs/images
C:/users/matt/../datasets/stop_signs/images/size_pad
```

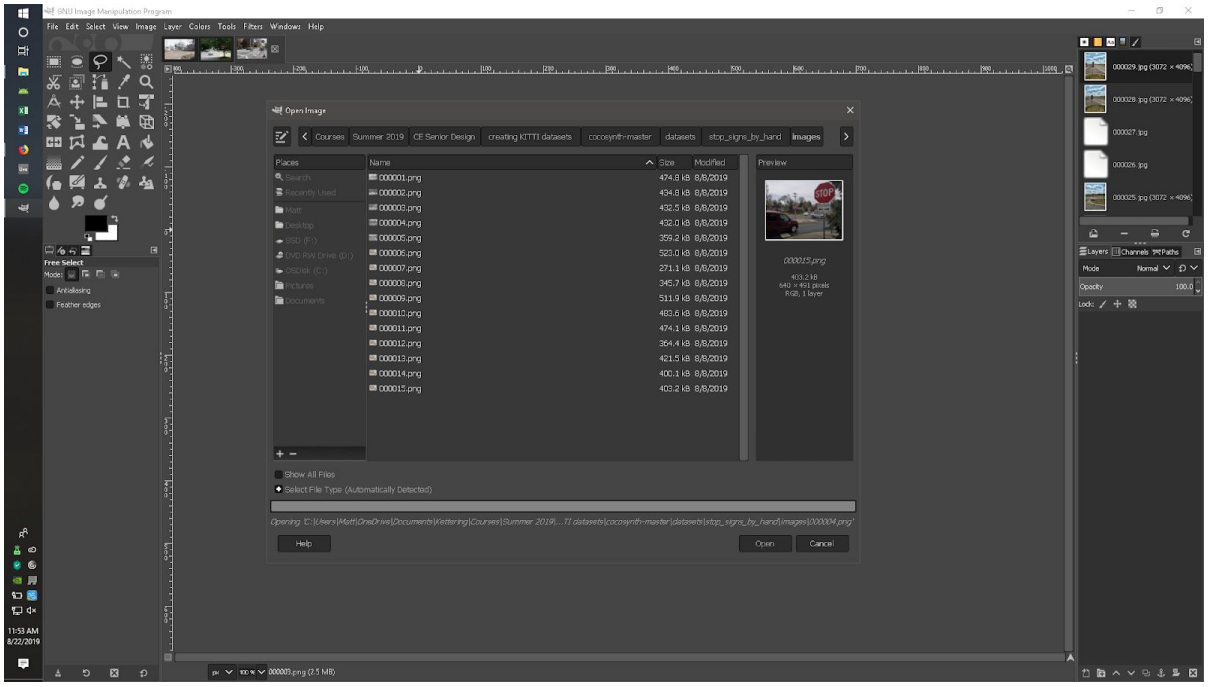
5. Dataset Creation with GIMP

In this section, we will hand annotate images of a dataset using GIMP to create our image masks. I do this using images of stop signs found on the internet, but feel free to use your own images.

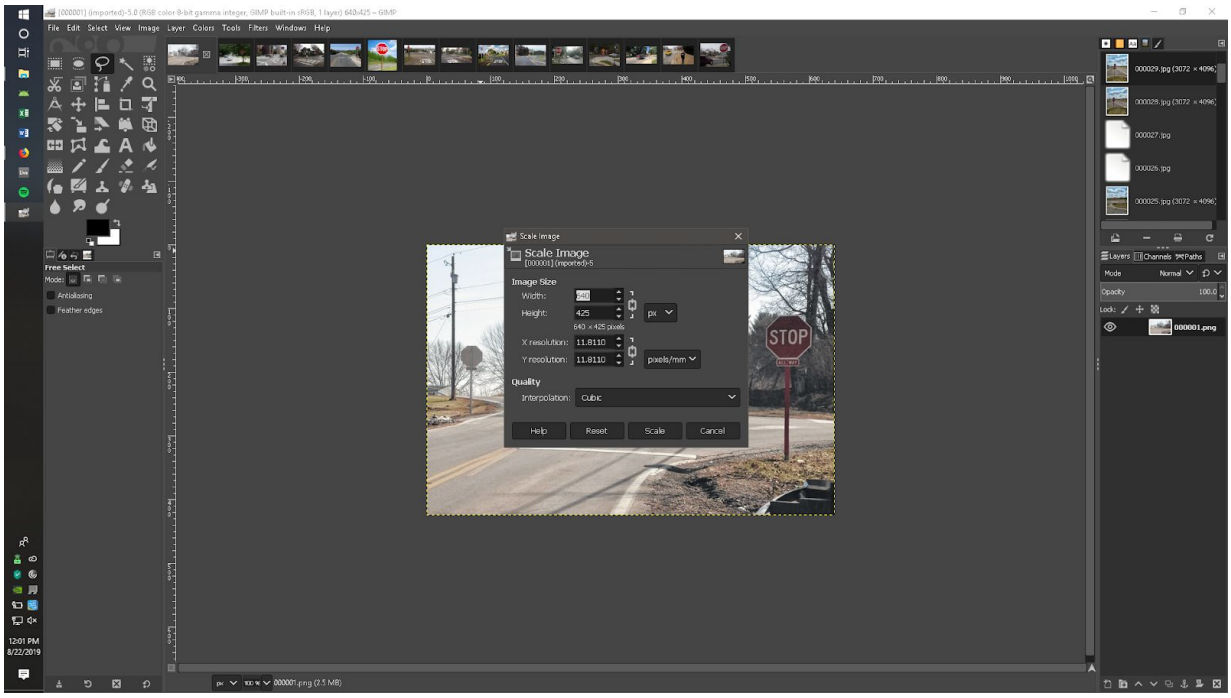
- ➢ Open GIMP > File > Open

Note: If, when you open GIMP, multiple windows open up - you can dock them all together by navigating to the windows tab > select single-window mode

- ➢ Navigate to Directory of Images > Select all you wish to edit



- Rescale all your images by navigating to the Image tab > Scale Image…
- Make sure to have your height linked to the width so that if you change one it will automatically scale the other.
- Change the height to 640 pixels, then select the scale button



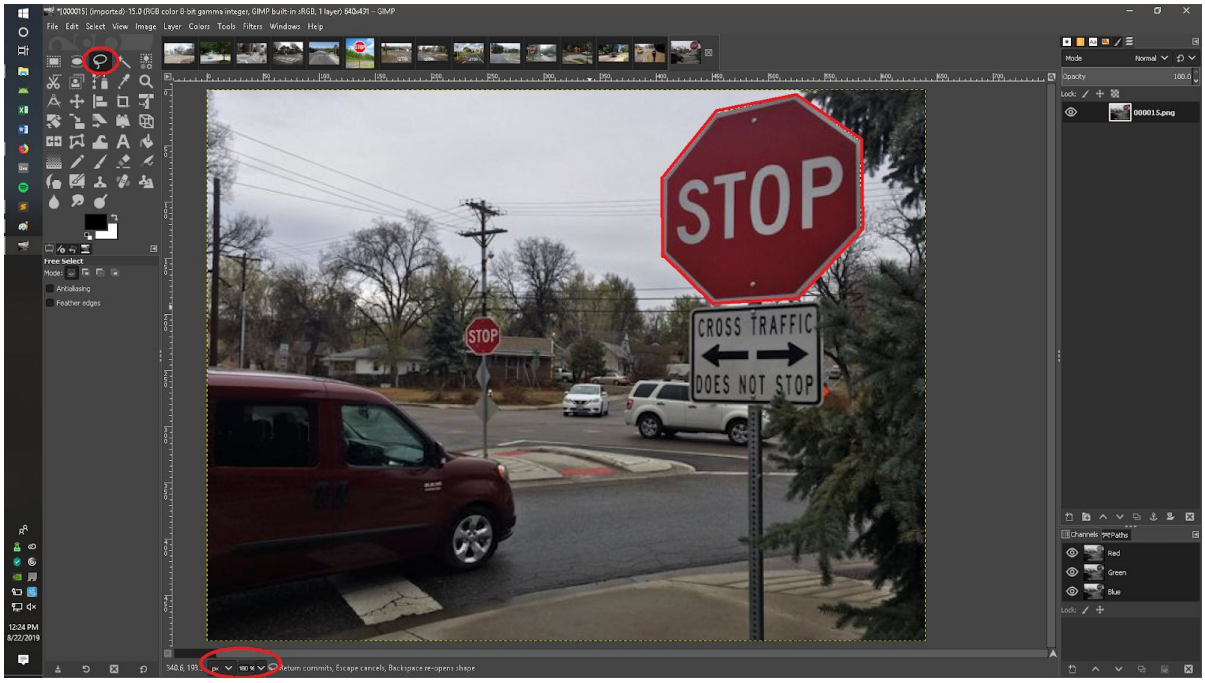
- ➢ Once you hit the scale button you should notice your image resize. Do this for all the images within your dataset

- ➢ Next export all of the resized images as .png files (lossless file type - google for clarification)

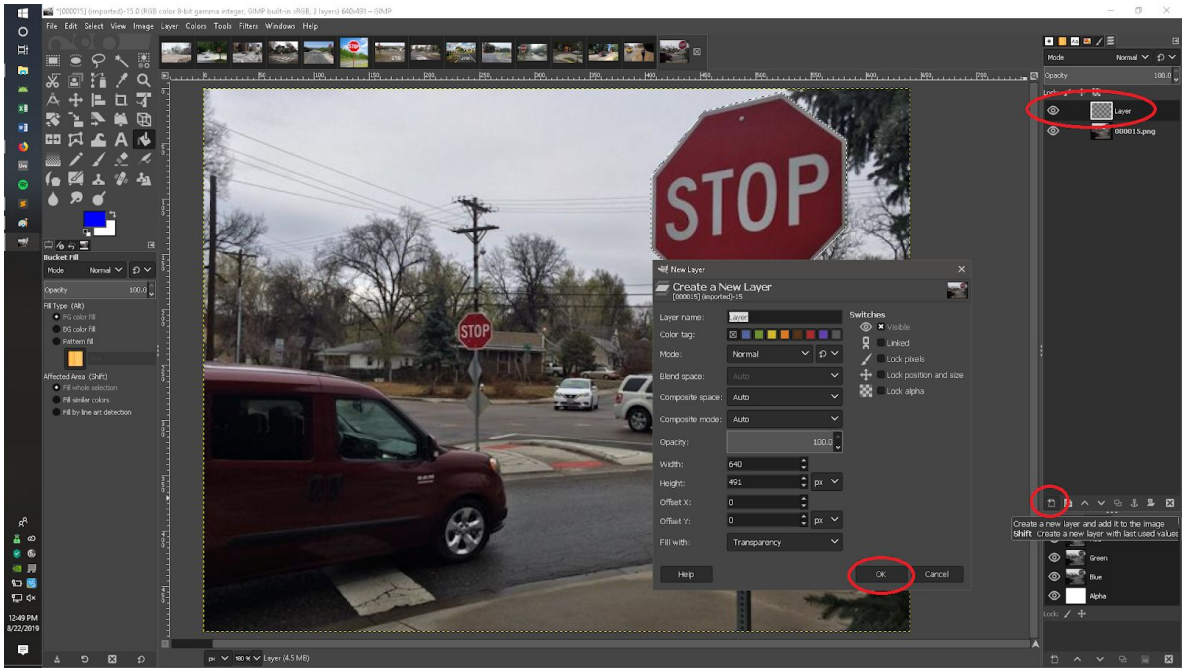
Note: You can overwrite all of your old images and/or delete any remaining jpeg files now. The naming scheme I used for my images was 000001.png, 000002.png, etc. - feel free to name it how you would like. Just know that we will be referencing these names in the mask_definitions.json file at a later time and it may be helpful to keep it a simple naming scheme.

- ➢ Next up, we will select our image with the lasso tool. This can be found on the left-hand side of the screen and is circled in the image below.

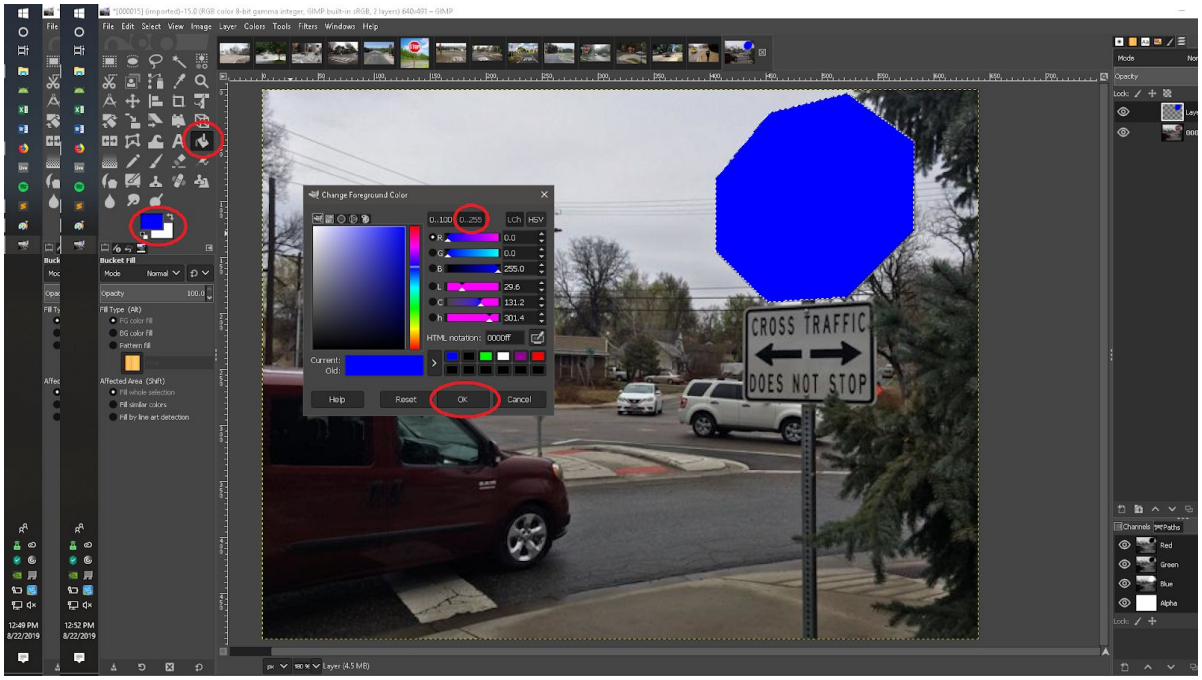
Note: It can be helpful to zoom to your object here. The zoom control is at the bottom of the screen and is also circled in red. Once you have the lasso tool selected and you are zoomed appropriately, click around the outside edges of your object. Try to get as close as possible to the edge. Notice that the edges of one of the stop signs in the image below is highlighted with red. This is where you should lasso your object.



- ➢ Once you have your object selected, add a new layer by selecting the small button at the bottom of the layers tab on the right hand side of the screen (shown below).



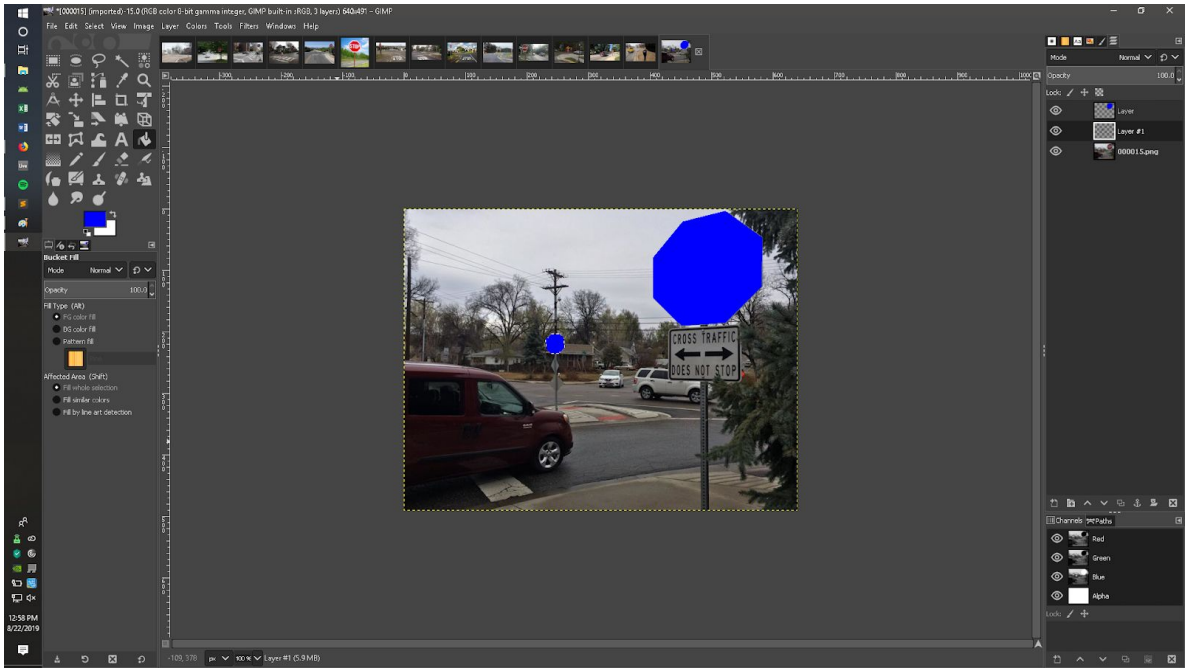
➢ Next, fill the selection on the new layer with a color by selecting the fill tool and changing the color to an RGB value of your choice. I chose Blue which has an RGB value of (0, 0, 255)



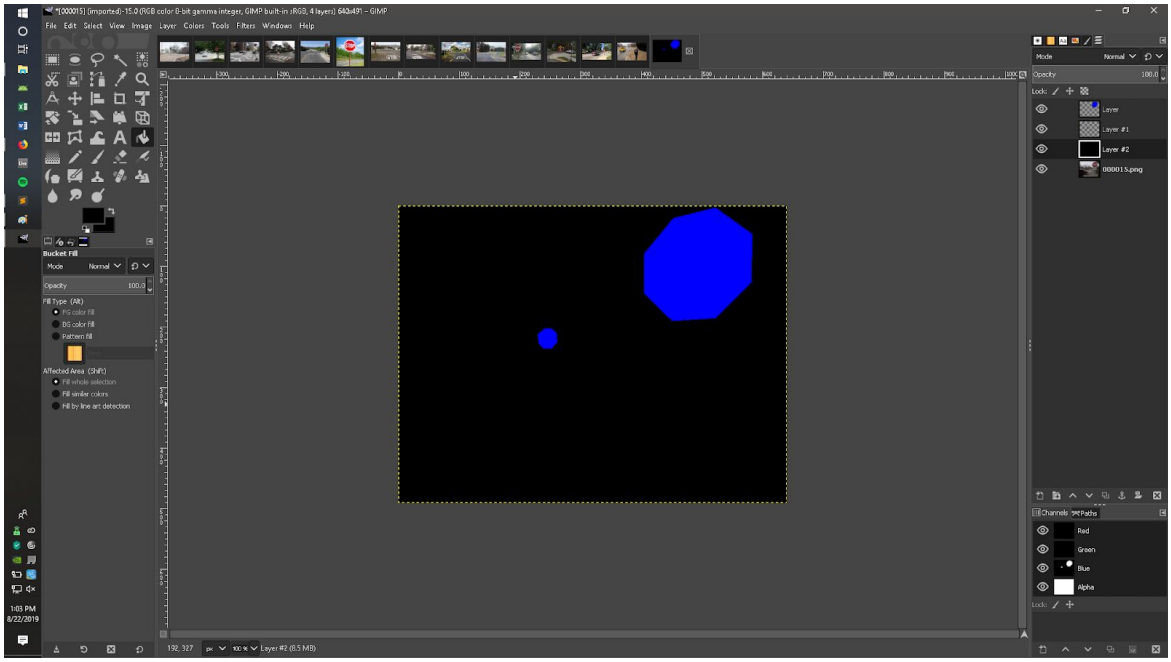
➢ Do this for each object you have in your image, ensuring that you add a new layer for each object. If you have multiple objects or if your object overlap each other, you will

want to use a different color mask for each object. However, if you have multiples of the same object, and they don't overlap, you can keep the mask the same color (shown below).

Note: To deselect your lasso selection, navigate to the Select tab > then click none.



➢ To wrap things up we create a final new layer and fill it in with black. Make sure that all of your object layers go before this final black layer.



- ➢ Export this image as a .png file and save it into a new folder named masks. Repeat these steps to make a mask for each image in your dataset.

6. Create Mask Definitions JSON file

The next step is to create the mask definitions JSON file which is required in order to create the COCO dataset. JSON files are essentially just made up of key-value pairs in a specific format. It is fairly straight-forward to follow.

Create a new text file within your dataset folder. Save it as mask_definitions.json.
Now edit the mask_definitions.json file to look something similar to this stop sign dataset's mask_definition.json file:

```
{
	"masks":
	{
		"images/000001.png":
		{
			"mask": "masks/000001.png",
			"color_categories":
			{
				"(0, 0, 255)": {"category": "stop_sign", "super_category": "null"}
			}
		},
		"images/000002.png":
		{
			"mask": "masks/000002.png",
			"color_categories":
			{
				"(0, 0, 255)": {"category": "stop_sign", "super_category": "null"}
			}
		},
		"images/000003.png":
		{
			"mask": "masks/000010.png",
			"color_categories":
			{
				"(0, 0, 255)": {"category": "stop_sign", "super_category": "null"},
				"(0, 255, 0)": {"category": "stop_sign", "super_category": "null"},
			}
		}
	},
	"super_categories":
```

```
	{
		"null": ["stop_sign"]
	}
}
```

For additional information regarding COCO's JSON format, visit their web page:
http://cocodataset.org/#format-data

7. Create Data Information JSON file

The last thing we need in order to generate the COCO annotations file is the dataset_info.json file. To do this first create a new text file within your dataset directory. Save it as dataset_info.json. Below is an example of what this file should contain. Don't be too worried about the content. This file is required to generate the COCO dataset annotations.

```
{
	"info":
	{
		"description": "Test dataset",
		"version": "1",
		"url": "no-url/datasets.com",
		"year": 2019,
		"contributor": "Matt Howlett",
		"date_created": "08/08/2019"
	},
	"license":{
		"url":"no-url/licenses.com",
		"id": 0,
		"name": "testing"
	}
}
```

8. COCO JSON Utils

The json_utils.py script will generate
Navigate to the cocosynth-master folder in the cmd prompt.

Run the coco_json_utils.py script in the cmd prompt with the following commands:

```
python ./python/coco_json_utils.py -md
./datasets/stop_signs_by_hand/mask_definitions.json -di
./datasets/stop_signs_by_hand/dataset_info.json
```

9. Convert to KITTI

The last thing we need to do to generate the KITTI formatted label text files is to run the coco2kitti.py script. Open this script within a text editor.

First things first, we have to make a few changes to the coco2kitti.py script. The primary change being the directory where it is grabbing our COCO JSON files. Scroll down to the bottom of the script and change lines 51, 52, and 53 to the correct data directory for where you saved your dataset. Note that this script expects the coco2kitti.py script to be within the annotations folder, be sure to move it there. Here is an example of the changes to be made to the coco2kitti.py script:

```
dataDir = 'C:/Users/Matt/CE Senior
Design/cocosynth-master/datasets/stop_signs_by_hand'
dataType = 'stop_signs'
annFile = '%s/annotations/coco_instances_%s.json' % (dataDir,
dataType)
```

After making the proper edits and saving the python script, run it by opening the command terminal. Navigate to the cocosynth-master directory within the command terminal and run the following command:

```
python ./datasets/stop_signs_by_hand/annotations/coco2kitti.py
```

Upon running this script, a new folder named labels should be within the cocosynth-master folder which contains a text file for each image/mask pair within your dataset.

10. KITTI dataset folder structure

Now that we have the label text files for each image, the last thing to do is to arrange the dataset with the correct folder structure. Here is how you should arrange your folder structure:

```
train/
├── images/
│   └── 000001.png
└── labels/
    └── 000001.txt
val/
├── images/
│   └── 000002.png
└── labels/
```

└── 000002.txt