Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Master of Science in Autonomous Systems

– Summer Semester 2018–

Semantic Segmentation using Resource Efficient Deep Learning

– Report on dataset creation –

by

Naresh Kumar Gurulingan
naresh.gurulingan@smail.inf.h-brs.de
Matr. no. 9030384

Contents

List of Tables

List of Figures

1 Overview of the dataset

Since semantic segmentation using deep learning is framed as a pixelwise classification task, an image of dimensions H×W×C requires a ground truth of dimensions H×W, where H and W are the height and width of the image in the dataset having C number of channels.

The scope of the dataset is to include objects associated to RoboCup @Work. The selected 18 objects are shown below:

Each of the objects were taken individually, placed on 3 different backgrounds and 30 images were taken. This lead to a total of 540 images which were to be manually labeled. Since, every pixel of the images needs to be labeled, the process of manual annotation would be time consuming. Therefore, a decision was made to first annotate the 540 images and later decide whether more images could be taken based on the effort required for annotation.

2 Selection of a labeling tool

In order to reduce the time required to annotate an image, it was imperative to select a tool which is specifically designed for semantic segmentation and also provides algorithms which helps the annotator by providing labeling automation to the highest possible extent.

The following available tools were evaluated for ease of use and time taken for annotation:

- LabelMe: web based tool is public and data would also be public.
- LabelMe Matlab toolbox: yet to try..
- University bonn annotation tool:
- Pixel annotation tool (using watershed algorithm): works in windows. Seems to be useful.
- Ratsnake: tool dint seem to be useful although the website had options like superpixel suggestions.
- LabelImg: Can be used but time consuming.
- Figi: used in medical image segmentation. Has many options. Still exploring.
- Supervisely.

- MATLAB ImageLabeler available in release R2017b (Computer Vision Toolbox).

3 Description of the labeling process

MATLAB ImageLabeler was used for the labeling process. At first, label definitions are created and exported to a .mat file. This file is used to load label definitions for all images to maintain consistency of labels. The contents of the .mat file is shown in the figure1.

labelDefs

19x4 table

	1 Name	2 Type	3 PixelLabelID	4 Description
1	'F20_20_B'	'4'	1	''
2	'S40_40_B'	'4'	2	''
3	'F20_20_G'	'4'	3	''
4	'S40_40_G'	'4'	4	''
5	'M20_100'	'4'	5	''
6	'M20'	'4'	6	''
7	'M30'	'4'	7	''
8	'R20'	'4'	8	''
9	'Bearing'	'4'	10	''
10	'Axis'	'4'	11	''
11	'DistanceTube'	'4'	12	''
12	'Motor'	'4'	13	''
13	'ContainerBlue'	'4'	14	''
14	'ContainerRed'	'4'	15	''
15	'BearingBoxAX01'	'4'	9	''
16	'BearingBoxAX16'	'4'	16	''
17	'EM_01'	'4'	17	''
18	'EM_02'	'4'	18	''
19	'Background'	'4'	19	''

Figure 1: Contents of the labelDefs .mat file

The ImageLabeler app, by default, provides different tools which help create pixel-wise labels2. These tools become accessible once an image and the label definitions are loaded. A short description of the tools is given below:

- Polygon: This can be used to trace an object boundary by placing dots. Once a closed contour is created, pixels within the contour get assigned the corresponding object label.

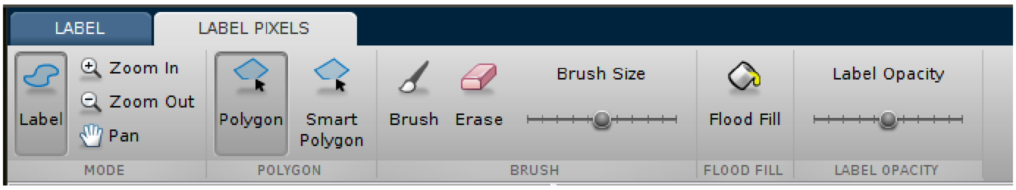

Figure 2: Tools provided by the ImageLabeler app

- Smart Polygon: Can be used in a similar fashion like the Polygon tool. This tool, in addition, tries to reach out to the nearby edges of the drawn polygon.
- Brush and Erase: Square shaped brush and eraser to either label a region or remove labels from a region. The size of the square can be changed by using the Brush Size slider.
- Flood Fill: This tool provides same labels to pixels which are similar in terms of the intensity with the selected pixel.
- Label Opacity: This tool provides a sliding bar which varies the opacity of the overlayed labels on the image. This is helpful to visualize the assigned labels.
- Zoom In, Zoom Out, Pan: These tools improve the ease of labeling by providing means to focus on particular regions by zooming and panning.

The ImageLabeler app by default assigns different colors to different objects to aid visualization. The label colors are shown in the ROI Label Definition window3.

The ImageLabeler app does not provide any tool to label all unlabeled pixels as background. In order to save time, each of the images taken, only have one object

In order to save time, the following workarounds have been used:

- The images taken for the dataset each have only one object in them.
- Only the object region is labeled.
- Since the ImageLabeler app does not provide any tool to label all unlabeled pixels as background, a python code which simply reads the label image and replaces unlabeled values 0 with background label value 19, was used for this purpose. The code is also used to double check the label image in order to avoid noisy labeling.

The Export Labels -> To File option can be used to save the annotations. This is done for all images individually to arrive at the folder structure shown in 4a:

The saved .mat file can be loaded into ImageLabeler again to further modify labels if required later. The 'Label_1.png' file located in the PixelLabelData folder (as can be

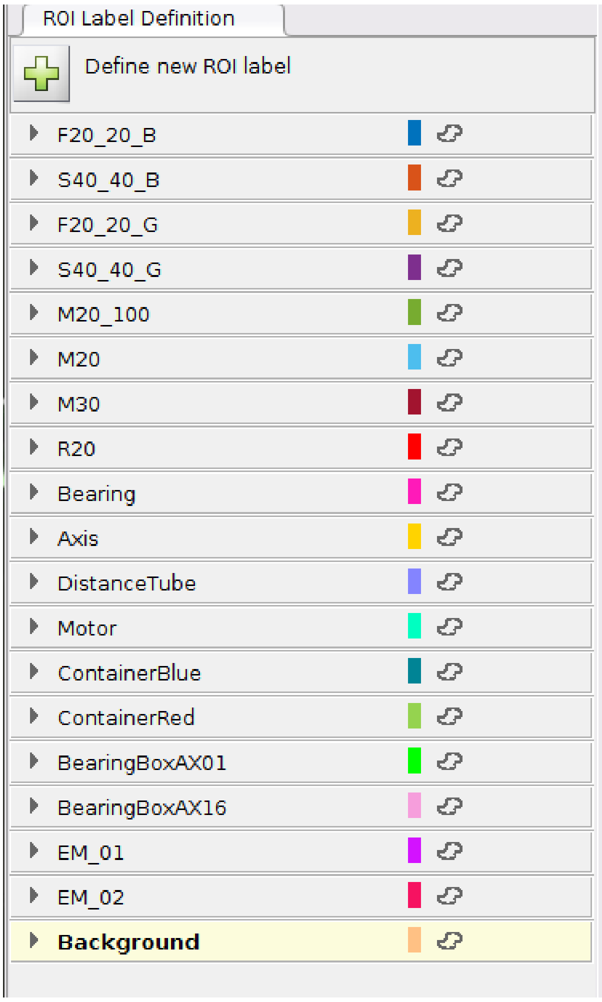

Figure 3: ROI Label Definitions window

seen in 4a) is the label image. This image is renamed to have the same name as the image file and the following folder structure is created by using a python code4b.

(a) Folder structure of saved labels

(b) Rearranged folder structure

Figure 4: Different folder structures

The final folder structure is shown in 5. The image folder and label folder are similar and contain object images and corresponding label images with same names.

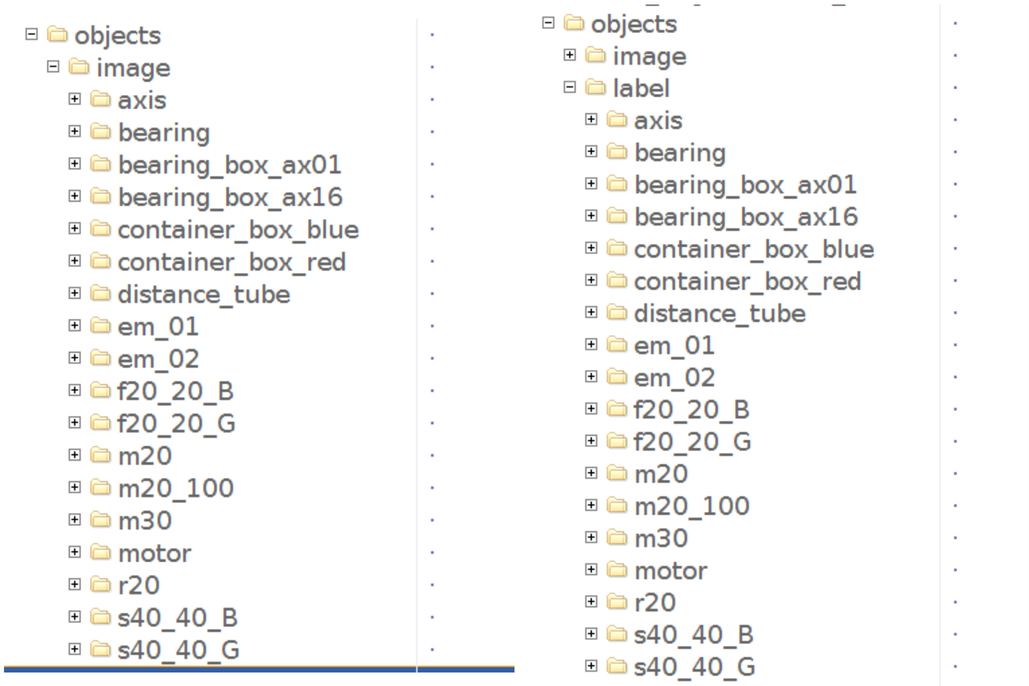

Figure 5: Folder structure showing different object folders in both image and label folders.

4 About the augmentation script

4.1 Motivation

- Manually labeling 540 images with the described process in 3 takes roughly 2160 minutes (roughly 4 minutes per image). This is equivalent to around 4 working days. Hence, creating a large dataset with manual labeling is not feasible.
- Taking images in a variety of real world backgrounds is also time consuming.
- Labeling images with multiple objects would take an even longer time.

These drawbacks could be overcome by randomly placing objects on a variety of different background images automatically using code.

4.2 Working

The following steps are performed to result in the generation of augmented images:

1 Given the path to the background images (for example, "./backgrounds'), the images in the path are read and stored in a list.

2 Given the path to the 'image' folder (for example, './objects/image') and the 'label' folder (for example, './objects/label'), the script fetches the paths of the label files and the corresponding paths of the image files. The number of label files available is also counted.

3 Each image and its corresponding label for each class of objects is read one by one. Different scales of the image and label for each image is also created based on the NUM_OF_SCALES keyword argument and added to a list. NUM_OF_SCALES argument can also be set to 'RANDAMIZE' in which case random number of scales in the range 1 to 5, will be used. If an added object is too small (determined using the MIN_OBJ_AREA_PERCENT argument) or too big (determined using the MAX_OBJ_AREA_PERCENT argument), it is removed from the list.

4 The list of objects contains the following details about each object:

   - 'obj_loc': The pixel locations in the image where the object is in the original image.
   - 'obj_vals': Intensity values of the object corresponding to 'obj_loc'.
   - 'label_vals': Label values of the object corresponding to 'obj_loc'.
   - 'obj_name': The name of the object.
   - 'rect_points': The top left and bottom right coordinates of the bounding rectangle of the object in pixel space.
   - 'obj_area': The area occupied by the object in pixel space.

   An example is shown in the figure6.

5 A list called 'augment_vector' is created with each element in the list containing the following details. Each of these elements denote an augmented image and determine how objects are placed on a background.

   - 'background_image': A randomly choosen background image. It is also made sure that each available background is used atleast once before reselecting a background.

  - 'num_objects_to_place': Number of objects to be placed in the current augmented image.
  - 'what_objects': A list of random numbers which determines what objects from the objects list is selected.
  - 'locations': A list of random locations in the pixel space where the selected objects need to be placed.

6 Each of the elements in the augment_vector is evaluated and elements whose objects occupy an area above MAX_OCCUPIED_AREA and whose object locations are too close to each other determined by MIN_DIST_BTW_LOCATIONS, are removed.

7 The elements in the augment_vector are taken one by one and based on the taken element, objects are placed on the background. The resulting augmented image in saved in './data_augmentation_results/image/' and ground truth is saved in './data_augmentation_results/ground_truth/'. Additionally object detection labels are also saved in csv files in the location './data_augmentation_results/obj_det/'. The plot function is provided to visualize the labels and visualized labels can also be saved in './data_augmentation_results/image_and_gt/' by setting save_data_preview to true.

```
{'obj_loc': array([[171, 284],
       [171, 285],
       [171, 286],
       ...,
       [242, 305],
       [242, 306],
       [242, 307]]), 'obj_vals': array([[ 70,  66,  68],
       [ 68,  63,  65],
       [ 69,  64,  67],
       ...,
       [120, 124, 123],
       [112, 114, 114],
       [112, 114, 114]], dtype=uint8), 'label_vals': array([1., 1., 1., ..., 1., 1., 1.]), 'obj_name': 'f20_20_B',
'rect_points': [171, 284, 242, 330], 'obj_area': 3266}
```

Figure 6: Details about an example object stored in the objects list

About the class and the object. Steps 1 to 4 in 4.2 are done by the class initialization and the rest using an object.

4.3 Keyword arguments

Details regarding the keyword arguments are provided in 1 and 2. Apart from these arguments, a label definition dictionary containing the object names and corresponding label values needs to be created and passed as argument to the class initialization.

Keyword argument	Description
IMAGE_DIMENSION	The dimensions of the images in the dataset can be changed using this parameter.
NUM_OF_SCALES	Determines the scales of objects to be created and added to the object list.
BACKGROUNDS_PATH	The path where the background files are located can be set using this argument.
IMAGE_PATH	The path where the image files are located can be set using this argument.
LABEL_PATH	The path where the label files are located can be set using this argument.
IMG_TYPE	The extension type of the image files can be provided here.
MIN_OBJ_AREA_PERCENT	The minimum area an object should occupy in the pixel space in percentage.
MAX_OBJ_AREA_PERCENT	The maximum area an object should occupy in the pixel space in percentage.
REMOVE_CLUTTER	If set to true, augmented images with too many objects and with objects which are too close to each other are removed.
NUM_OF_IMAGES	The number of augmented images required to be generated can be set here.
MAX_OBJECTS_PER_IMAGE	The maximum number of objects which are allowed in an augmented image.
GENERATION_REATTEMPTS	The number of reattempts which can be made to regenerate removed augment vectors.
CLEAR_AUGMENT_VECTOR	Can be used to clear generated augment vectors.
MIN_DIST_BTW_LOCATIONS	Minimum distance in terms of number of pixels which can be allowed between two random locations in an augment vector.
MAX_OCCUPIED_AREA	The maximum area which can be occupied by all the objects in the augment vector.
PREVIEW_DATA	If set to true, the generated objects and labels are displayed inline in the notebook.
SAVE_DATA_PREVIEW	If set to true, the generated visualization plot for image and label is saved.
GET_OBJ_DET_LABEL	If set to true, object detection labels are saved.

Table 1: Keyword Arguments Description

4.4 Sample results

Some sample results can be seen in 7. The yellow bounding box represents the object detection label and the different colors of the segmentation labels denote different label values.

Keyword argument	Default value	DataAugmentation class argument of method
IMAGE_DIMENSION	[480, 640]	__init__ method
NUM_OF_SCALES	2	__init__ method
BACKGROUNDS_PATH	'./backgrounds'	__init__ method
IMAGE_PATH	'./objects/image'	__init__ method
LABEL_PATH	'./objects/label'	__init__ method
IMG_TYPE	'.jpg'	__init__ method
MIN_OBJ_AREA_PERCENT	0.3	__init__ method
MAX_OBJ_AREA_PERCENT	70	__init__ method
REMOVE_CLUTTER	True	create_augment_vector method
NUM_OF_IMAGES	50	create_augment_vector method
MAX_OBJECTS_PER_IMAGE	6	create_augment_vector method
GENERATION_REATTEMPTS	100	create_augment_vector method
CLEAR_AUGMENT_VECTOR	True	create_augment_vector method
MIN_DIST_BTW_LOCATIONS	70	create_augment_vector method
MAX_OCCUPIED_AREA	0.8	create_augment_vector method
PREVIEW_DATA	False	perform_augmentation method
SAVE_DATA_PREVIEW	False	perform_augmentation method
GET_OBJ_DET_LABEL	True	perform_augmentation method

Table 2: Details regarding the keyword arguments

5 Meta-data of the dataset

6 Conclusion and possible directions of improvement

- Improve the way in which PixelLabelData is saved.
- Integrating 'rest of the pixels are background' into MATLAB ImageLabeler.
- Integration of augmentation script with MATLAB ImageLabeler.
- GUI for the augmentation script.

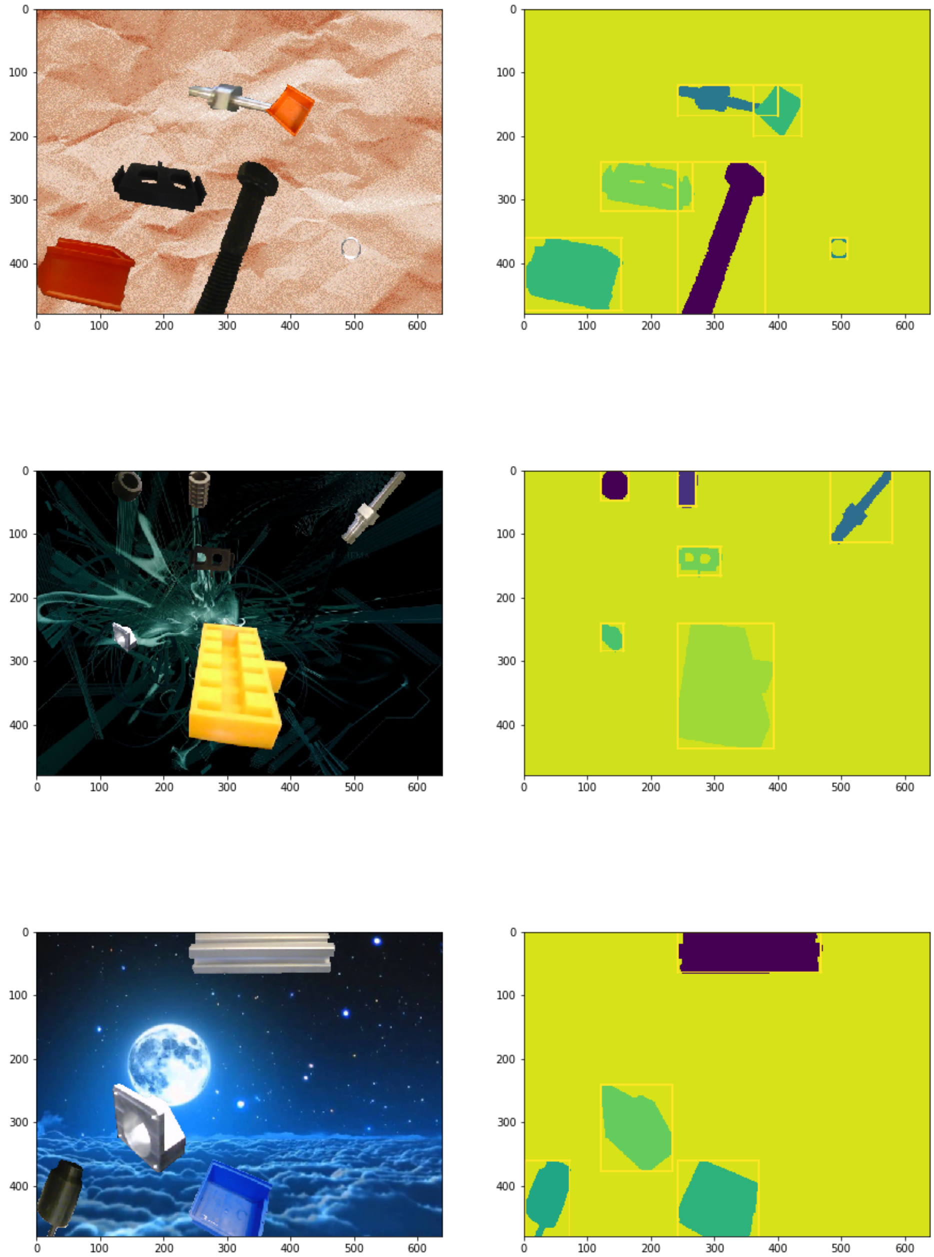

Figure 7: Sample results produced by the augmentation script

References

[1]