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.

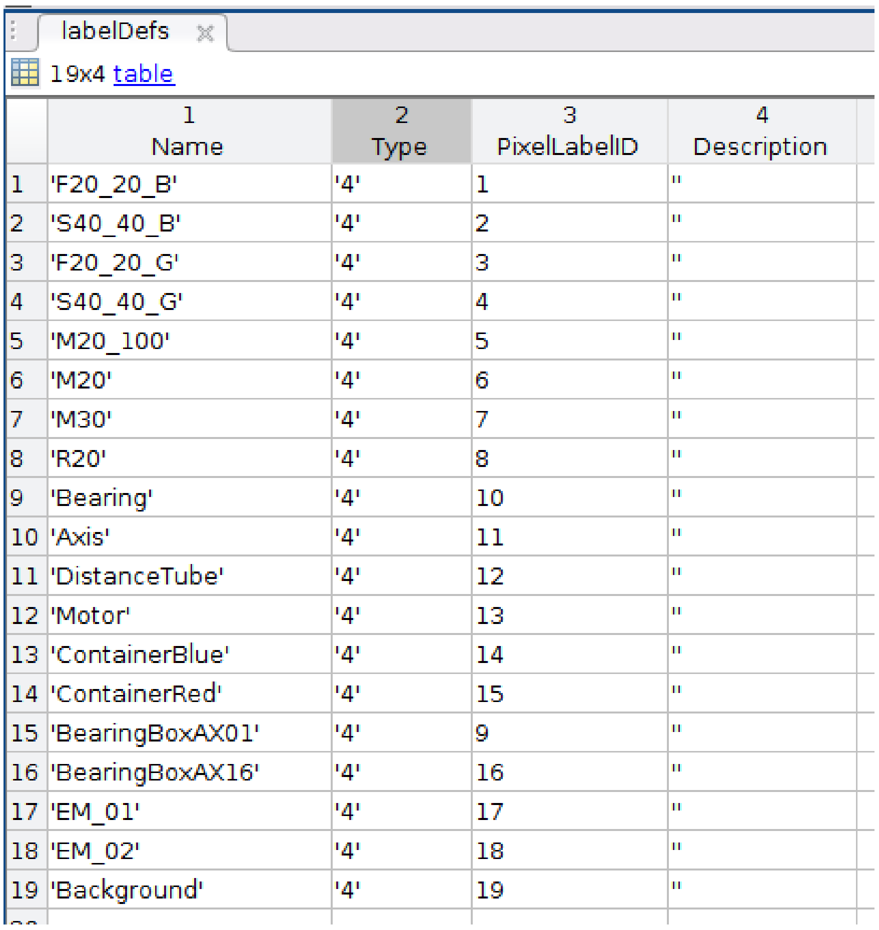
The following available tools were evaluated for ease of use and time taken for annotation:

- 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.

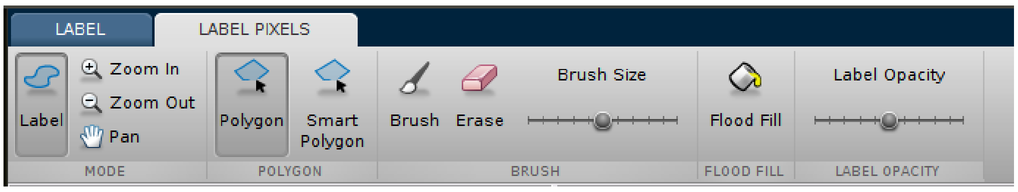


Figure 2: Tools provided by the ImageLabeler app

- 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.

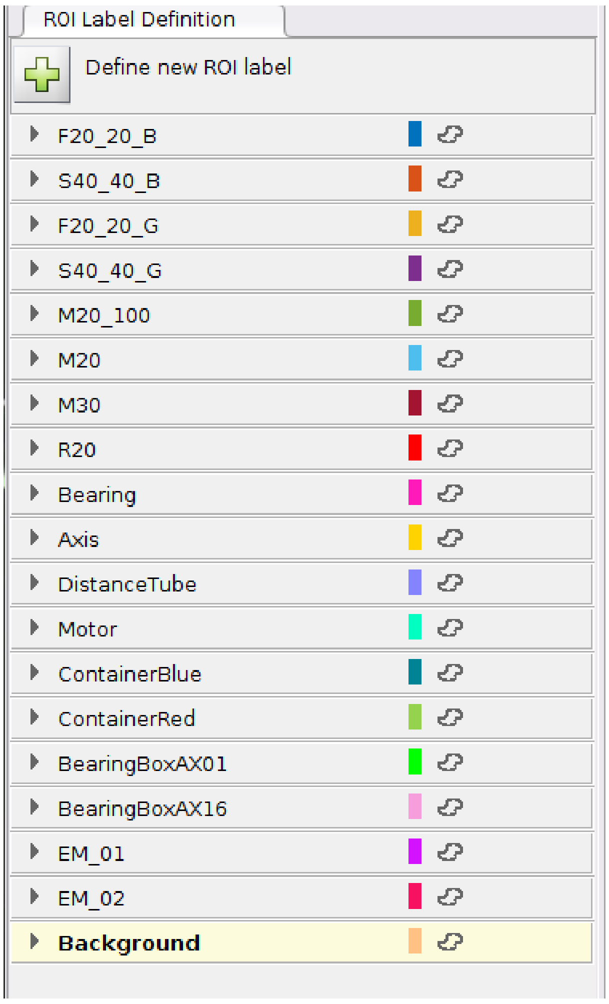
The ImageLabeler app does not provide any tool to label all unlabeled pixels as background. In order to save time, each of the images taken, only have one object

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

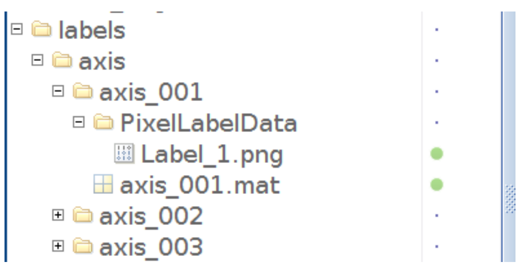

ROI Label Definition

Define new ROI label

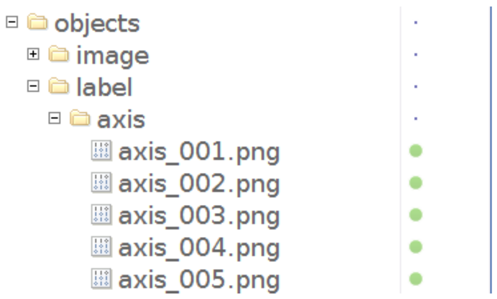
F20_20_B
S40_40_B
F20_20_G
S40_40_G
M20_100
M20
M30
R20
Bearing
Axis
DistanceTube
Motor
ContainerBlue
ContainerRed
BearingBoxAX01
BearingBoxAX16
EM_01
EM_02
Background

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.


```
labels
  axis
    axis_001
      PixelLabelData
        Label_1.png
      axis_001.mat
    axis_002
    axis_003
```

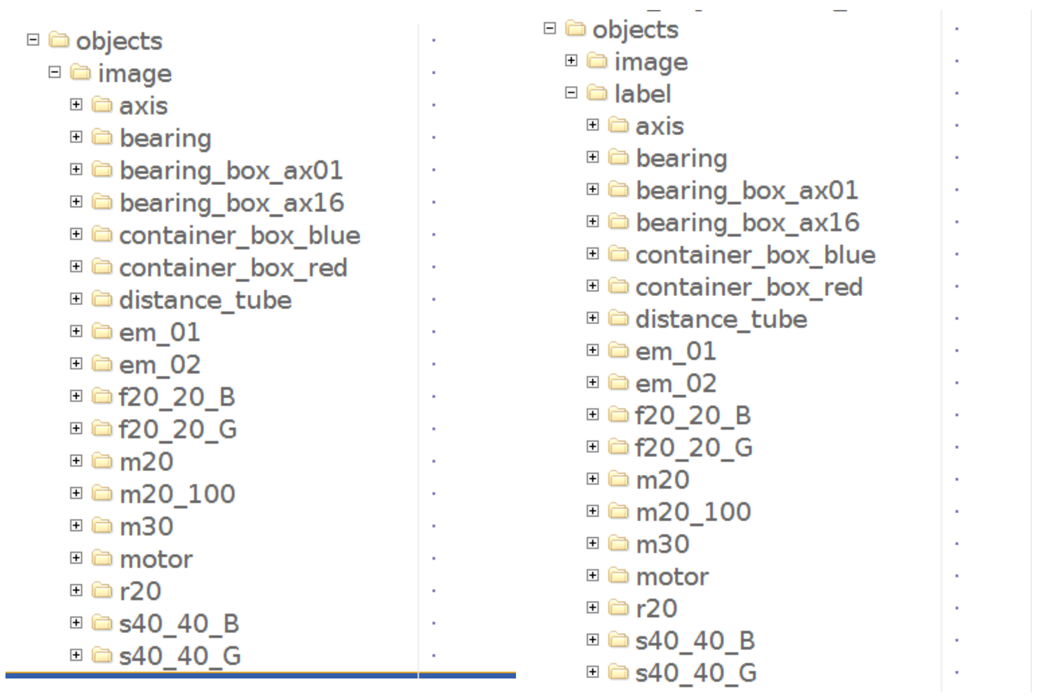
(a) Folder structure of saved labels


```
objects
  image
  label
    axis
      axis_001.png
      axis_002.png
      axis_003.png
      axis_004.png
      axis_005.png
```

(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.



```
⊟ objects
  ⊟ image
    ⊞ axis
    ⊞ bearing
    ⊞ bearing_box_ax01
    ⊞ bearing_box_ax16
    ⊞ container_box_blue
    ⊞ container_box_red
    ⊞ distance_tube
    ⊞ em_01
    ⊞ em_02
    ⊞ f20_20_B
    ⊞ f20_20_G
    ⊞ m20
    ⊞ m20_100
    ⊞ m30
    ⊞ motor
    ⊞ r20
    ⊞ s40_40_B
    ⊞ s40_40_G
```

```
⊟ objects
  ⊞ image
  ⊟ label
    ⊞ axis
    ⊞ bearing
    ⊞ bearing_box_ax01
    ⊞ bearing_box_ax16
    ⊞ container_box_blue
    ⊞ container_box_red
    ⊞ distance_tube
    ⊞ em_01
    ⊞ em_02
    ⊞ f20_20_B
    ⊞ f20_20_G
    ⊞ m20
    ⊞ m20_100
    ⊞ m30
    ⊞ motor
    ⊞ r20
    ⊞ s40_40_B
    ⊞ s40_40_G
```

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.

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.

References

[1]