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

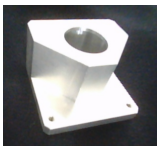
Since semantic segmentation using deep learning is framed as a pixelwise classification task, an image of dimensions H×W×C requires a ground truth of dimensions H×W, where H and W are the height and width of the image in the dataset having C number of channels.

The scope of the dataset is to include objects associated to RoboCup @Work. The selected 18 objects are shown in 1.

(a) axis [1]

(b) bearing [1]

(c) bearing box AX01 [1]

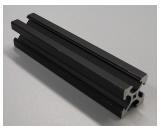


(d) bearing box AX16

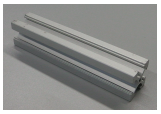
(e) container blue [1]

(f) container red [1]

(g) distance tube [1]



(h) F20_20_B [1]



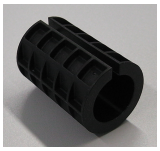
(i) F20_20_G [1]

(j) M20 [1]

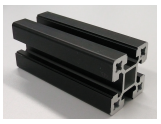
(k) M20_100 [1]

(l) M30 [1]

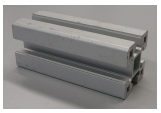
(m) motor [1]



(n) R20 [1]



(o) S40_40_B [1]



(p) S40_40_G [1]

(q) em_01

(r) em_02

Figure 1: Different objects required in the dataset

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).

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 2: Contents of the labelDefs .mat file

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 figure2.

The ImageLabeler app, by default, provides different tools which help create pixel-wise labels3. 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.
- 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.

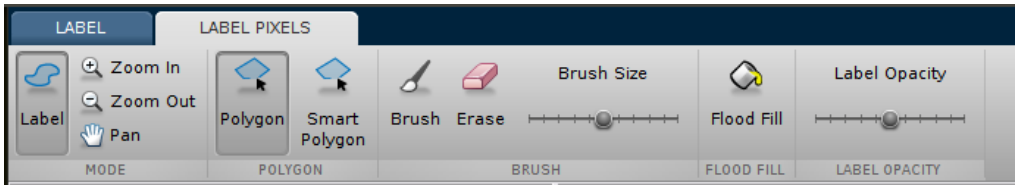


Figure 3: Tools provided by the ImageLabeler app

- 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 window4.

The ImageLabeler app does not provide any tool to label all unlabeled pixels as background. 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 5a.

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 seen in 5a) is the label image. This image is renamed to have the same name as the image file and a folder structure as in 5b is created by using a python code.

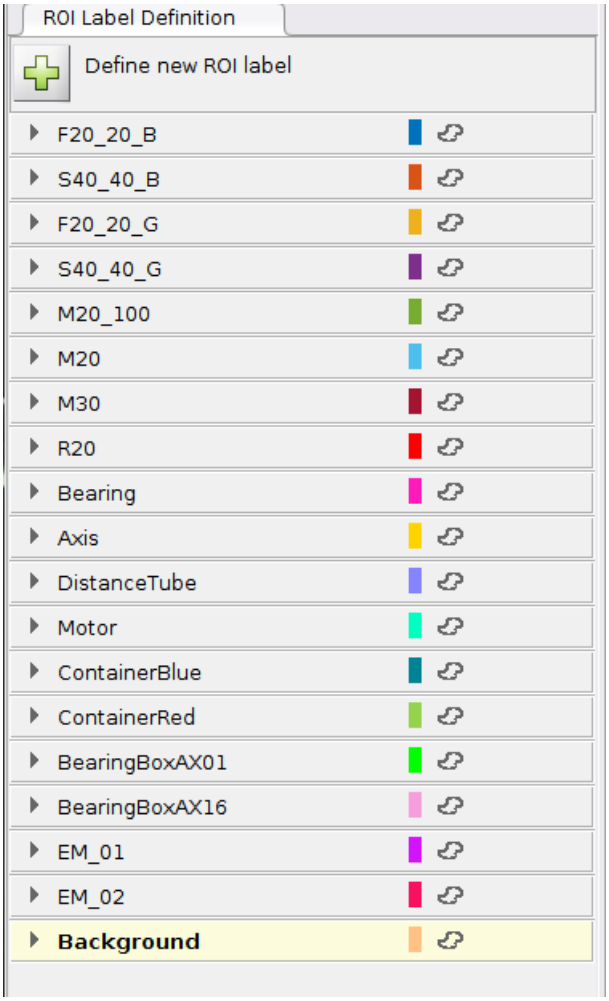


Figure 4: ROI Label Definitions window

(a) Folder structure of saved labels

(b) Rearranged folder structure

Figure 5: Different folder structures

The final folder structure is shown in 6. The image folder and label folder are similar and contain object images and corresponding label images with same names.

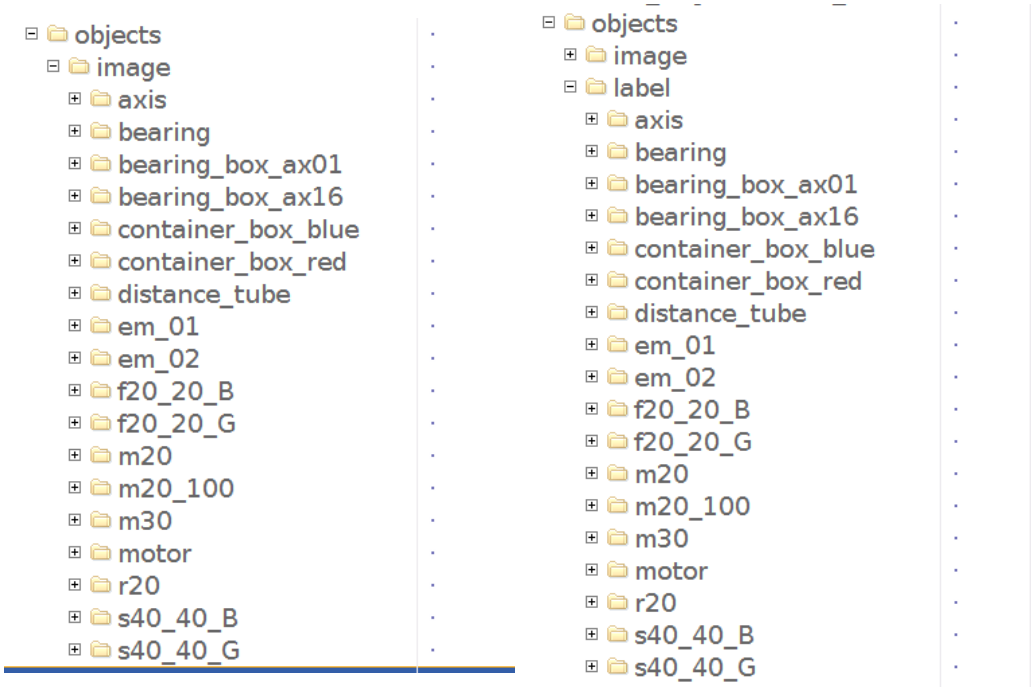


Figure 6: Folder structure showing different object folders in both image and label folders.

4 About the artificial image generation algorithm

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 an algorithm.

4.2 Working

The artificial image generation algorithm requires that every image provided must have just one object. The entire process of artificial image generation can be divided into 6 broad steps:

1 **External interface**: An interface to obtain possible parameters to control the generation process. These parameters are obtaiend through argparse command line GUI and are called generator options.

2 **Get backgrounds and data**: Fetch all the backgrouds, images and corresponding labels from the provided respective background, image and label paths. First, all the images in the backgrounds path are read. Then, all available images in the label path is read and the corresponding image files in the provided image path is read. The images in the image path must have the same name as that of the corresponding labels but can be of a different format. The default image format is ".jpg".

3 **Get object details**: Fetch details regarding every object and its different scales. The details include information regarding object locations in an image, object values, label values, the object name, points in pixel space denoting a bounding rectangle around the object, and object area. The scales for every object is determined at random. Scaled objects which are too small or too big as determined by generator options are removed.

4 **Generate augmenter list**: Every element in the augmenter list denotes an artificial image and contains information including the chosen background image, the number of objects to place in the artificial image, which objects from the object details list are selected and locations in pixel space where the selected objects need to be placed. In this stage, elements which are cluttered with too many objects are removed as determined by the generator options.

5 **Generate artificial images**: Based on every element in the augmenter list, artificial images and corresponding labels are generated. Every element is taken one by one. The selected objects are placed on the selected background in the corresponding specified location, one by one. The resultant artificial image and semantic label is saved in the directory specified using generator options. Additionally, object detection labels, semantic masks and visualization previews can be saved by configuring generator options.

6 **Generate preview**: The generated images and labels are visualized to verify the generation process.

4.3 Generator options

A number of arguments can be configured to control the generation process. Configuration of generator options is possible through command line GUI. Details regarding the arguments are provided in 1 and 2.

Generator options	Description
image_dimension	Dimension of the real images.
num_scales	Number of scales including original object scale.
backgrounds_path	Path to directory where the background images are located.
image_path	Path to directory where real images are located.
label_path	Path to directory where labels are located.
real_img_type	The format of the real image.
min_obj_area	Minimum area in percentage allowed for an object in image space.
max_obj_area	Maximum area in percentage allowed for an object in image space.
save_label_preview	Save image+label in single image for preview.
save_obj_det_label	Save object detection labels in csv files.
save_mask	Save images showing the segmentation mask.
image_save_path	Path where the generated artificial image needs to be saved.
label_save_path	Path where the generated segmentation label needs to be saved.
preview_save_path	Path where object detection labels needs to be saved.
obj_det_save_path	Path where object detection labels needs to be saved.
mask_save_path	Path where segmentation masks needs to be saved.
start_index	Index from which image and label names should start.
name_format	The format for image file names.
remove_clutter	Remove images cluttered with objects.
num_images	Number of artificial images to generate.
max_objects	Maximum number of objects allowed in an image.
num_regenerate	Number of regeneration attempts of removed details dict.
min_distance	Minimum pixel distance required between two objects.
max_occupied_area	Maximum object occupancy area allowed.
scale_ranges	Can be used to change the zoom range of specific objects.

Table 1: Description of generator options

4.4 Sample results

Sample results of the artificial image generation algorithm can be seen in 10. The bounding box represents the object detection label and the different colors of the segmentation labels denote different label values.

Generator options	Default value	Is required?
image_dimension	[480, 640]	Not required
num_scales	'randomize'	Not required
backgrounds_path	-	Required
image_path	-	Required
label_path	-	Required
real_img_type	'.jpg'	Not required
min_obj_area	20	Not required
max_obj_area	70	Not required
save_label_preview	False	Not required
save_obj_det_label	False	Not required
save_mask	False	Not required
image_save_path	-	Required
label_save_path	-	Required
preview_save_path	None	Required if save_label_preview is True
obj_det_save_path	None	Required if save_obj_det_label is True
mask_save_path	None	Required if save_mask is True
start_index	0	Not required
name_format	'%05d'	Not required
remove_clutter	True	Not required
num_images	20	Not required
max_objects	10	Not required
num_regenerate	100	Not required
min_distance	100	Not required
max_occupied_area	0.8	Not required
scale_ranges	None	Not required

Table 2: Default value of generator options and whether the options are required to be set.

4.5 Notable features of the artificial image generator

4.6 Artificial images for each dataset split

The real images are split into training, validation and test sets. Real images in each of these sets are used to generate artificial images for the coresponding set. This ensures that the final training, validation and test sets are different from each other. The scale range of distance_tube was set to 1.1 to 2.0 for generating training artificial images and to 0.6 to 1.2 for generating validation and test artificial images. The reason for this setting is provided in the data analysis section 5.2.

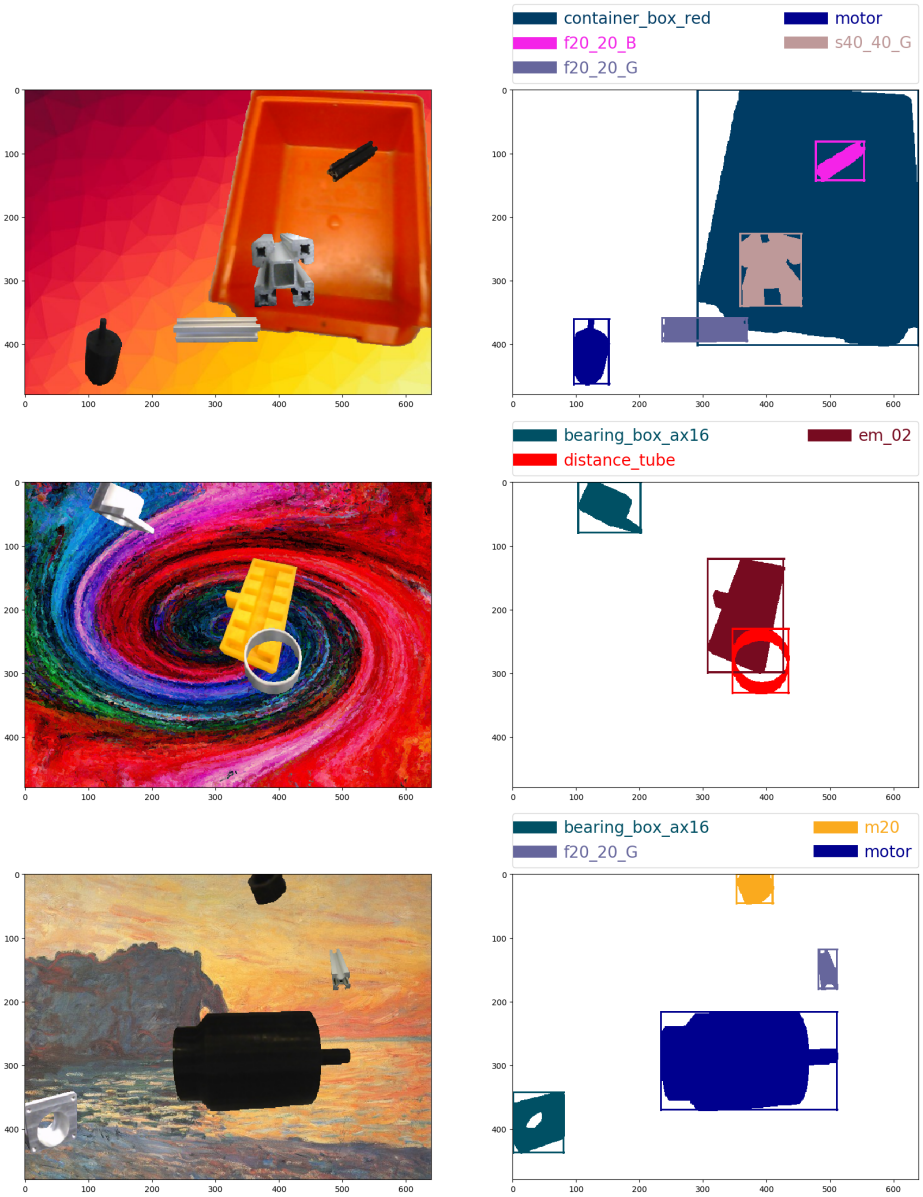


Figure 7: Sample results produced by the artificial image generation algorithm. In each row, the image on the left shows the generated artificial image and the image on the right shows a visualization of the semantic segmentation label and object detection label. At the top of every label visualization image, the objects in the image and their corresponding colors in the visualization are indicated.

5 Creation of dataset variants:

Different variants of the dataset are created based on the properties of the objects in the dataset, and the type of background images used for generation of artificial images.

5.1 Motivation

Looking into the objects present in the dataset, it is apparant that some objects are similar in certain aspects. For instance, the objects m20 and m30 are very similar to each other except that m30 is bigger in size and has a slightly different color. Because of the similarities existing among objects, the segmentation model could face certain difficulties as listed below:

- **Inability to distinguish size**: The segmentation model is given no information regarding the positions of the camera or the object in the real world. If camera extrinsic calibration information is available to the segmentation model, the model could possibly learn to distinguish different sizes. However, such information is not available. In addition, the objects in the artificial images are randomly scaled to different sizes thereby removing any size related information available.
- **Inability to distinguish subtle variations in color**: The real images were taken under different lighting conditions. As a result, there is no consistent difference in color information available between classes. This makes it difficult for the segmentation model to learn patterns in color information.
- **Inability to distinguish shapes**: Certain objects are closly related to each other in terms of shape and differ only slightly. For instance, bearing_box_ax16 and bearing_box_ax01 are similar in shape except in a few viewpoints as illustrated in 8 and 9. In such cases, in certain viewpoints, the segmentation model would not be able to distinguish between similarly shaped objects.

(a) A viewpoint of bearing box ax01

(b) A viewpoint of bearing box ax16

Figure 8: A similar viewpoint of bearing box ax01 and bearing box ax16 where the difference in shapes between the two objects is clearly visible.

5.2 Dataset variants

Four different dataset variants have been created as listed below:

(a) A viewpoint of bearing box ax01

(b) A viewpoint of bearing box ax16

Figure 9: A similar viewpoint of bearing box ax01 and bearing box ax16 where the appear similar in shape.

- **atWork_full**: This variant has different label values for all the 18 objects in the dataset and an additional label value for background. The total number of classes is 19 including background and the label values range from 0 to 18. The different classes in this variant along with their label values are listed in 3.

- **atWork_size_invariant**: In this variant, objects which are similar to each other in terms of shape but differ in size are combined together into one class. On this regard, f20_20_B and s40_40_B are combined and named f_s20_40_20_40_B. Similarly, f20_20_G and s40_40_G are combined and named f_s20_40_20_40_G. m20 and m30 are combined and named m20_30. The two bearing boxes, bearing_box_ax01 and bearing_box_ax16 are also combined together as they are similar to each other in certain viewpoints. They form the new class bearing_box. The objects in this variant along with their label values are listed in 7. This variant is named "atWork_size_invariant" as in this variant, the major change deals with the ignorance of the size of the objects as distinguishing information.

- **atWork_similar_shapes**: In the previous variant "atWork_size_invariant", objects similar in terms of shape but different in terms of color were treated as separate classes. In this variant, variation in terms of color is also ignored. In addition to the previous variant, f_s20_40_20_40_B and f_s20_40_20_40_G are combined and named f_s20_40_20_40_B_G. The container boxes, container_box_red and container_box_blue were also combined to form the new class container_box. This variant is named "atWork_similar_shapes" as objects with similar shapes are given equal label values. Details regarding this variant are listed in 5.

- **atWork_binary**: An interesting question would be, "how would a segmentation model perform when it is just tasked with segmenting foreground from background". To address this question, an additional variant is created called "atWork_binary" where are the objects of interest are combined to form the "foreground" class with label value 1. The "background" class retains its label value of 0. The classes in this variant are listed in 6.

Variant Name	Object Name	Label Value
atWork_full	background	0
	f20_20_B	1
	s40_40_B	2
	f20_20_G	3
	s40_40_G	4
	m20_100	5
	m20	6
	m30	7
	r20	8
	bearing_box_ax01	9
	bearing	10
	axis	11
	distance_tube	12
	motor	13
	container_box_blue	14
	container_box_red	15
	bearing_box_ax16	16
	em_01	17
	em_02	18

Table 3: Details of the "atWork_full" variant

Variant Name	Object Name	Label Value
atWork_size_invariant	background	0
	f_s20_40_20_40_B	1
	f_s20_40_20_40_G	2
	m20_100	3
	m20_30	4
	r20	5
	bearing_box	6
	bearing	7
	axis	8
	distance_tube	9
	motor	10
	container_box_blue	11
	container_box_red	12
	em_01	13
	em_02	14

Table 4: Details of the "atWork_size_invariant" variant

5.3 The shades of white dataset

The backgrounds used for the artificial image generation process are images with a variety of different colors, textures and so on. In essense, the background images do not seem to follow any pattern as such. As a result, the generated artificial images are un-

Variant Name	Object Name	Label Value
atWork_similar_shapes	background	0
	f_s20_40_20_40_B_G	1
	m20_100	2
	m20_30	3
	r20	4
	bearing_box	5
	bearing	6
	axis	7
	distance_tube	8
	motor	9
	container	10
	em_01	11
	em_02	12

Table 5: Details of the "atWork_similar_shapes" variant

Variant Name	Object Name	Label Value
atWork_binary	background	0
	foreground	1

Table 6: Details of the "atWork_binary" variant

likely to be similar to an image taken by an atWork robot. In order to address this, the background images used in the artificial image generation process are all replaced with images which mostly contain shades of white color in them. With this as the only change, the entire artificial image generation process and variant creation process is repeated to arrive at a new dataset which is named "shades of white".

6 Data analysis:

All the variants of the dataset are analyzed in terms of the pixels occupied by each class in percentage and the number of images in which each classs appears.

6.1 Analysing the "atWork_full" variant:

7 Meta-data of the dataset

Details regarding the dataset is provided in table 3.

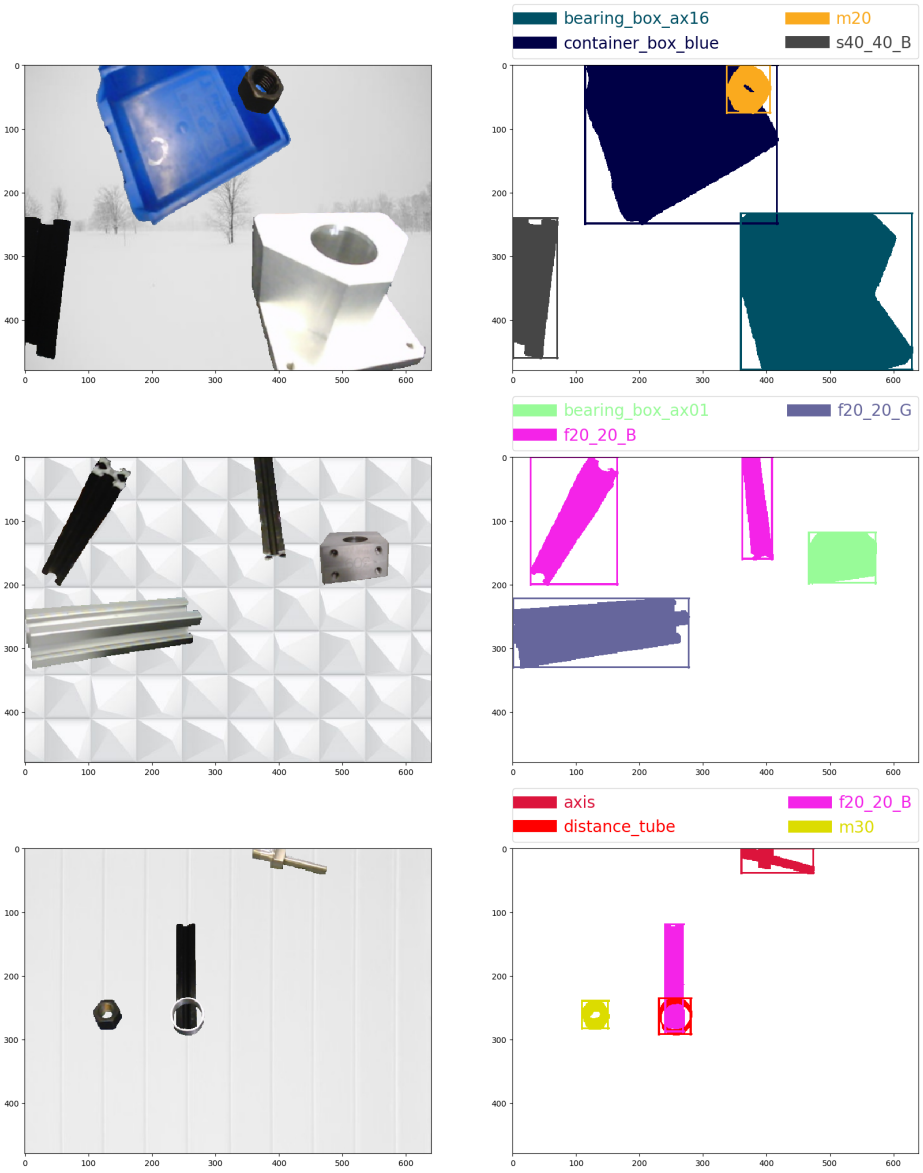


Figure 10: Sample results produced by the artificial image generation algorithm for the "shades of white" dataset.

	Training	Validation	Test
Real Images	30 per object. Total: 30×18=540	5 per object. Total: 5×18=90	5 per object. Total: 5×18=90
Augmented Images	5000	810	810
Total Images	5540	900	900

Table 7: Meta-data of the dataset

8 Conclusion and possible directions of improvement

Creating a custom dataset for a desired application is evidently challenging. To overcome the time consuming nature of creating annotations for semantic segmentation, choices such as 1. placing just 1 object per image while taking real images and 2. augmenting the objects on a random selection of diverse backgrounds, were made. This method of augmentation, although inspired by dataset generation method used in [2] and the Synthia dataset [3], takes a different approach. Unlike [2], which uses 3D CAD models, this approach does not require any 3D models. Also, this approach does not require a virtual world as used by the Synthia dataset [3]. However, the effectiveness of the dataset is yet to be verified by training and testing available segmentation models. The following list provides possible directions of improvement:

- The ImageLabeler app by default saves the label '.png' file with the name 'Label_1.png' in a folder called PixelLabelData. A automation script can be written and added to the ImageLabeler to provide options to save the label file in a way the user wants.
- Creating a way to replace all unlabeled pixels with the label value of 'background' from within the ImageLabeler would be helpful. For now, this is done by first exporting the label, then loading the label using opencv in python to replace 0 (value of unlabeled pixels) with 19 (value of 'background').
- The augmentation script is written in python and is independent of the MATLAB ImageLabeler app. This can be improved by including a way to start augmentation right from the ImageLabeler.

References

[1] github: robocup-at-work/rulebook/images; link: https://github.com/robocup-at-work/rulebook/tree/master/images.

[2] Param S. Rajpura, Manik Goyal, Hristo Bojinov, and Ravi S. Hegde. Dataset augmentation with synthetic images improves semantic segmentation. *CoRR*, abs/1709.00849, 2017.

[3] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio Lopez. The SYNTHIA Dataset: A large collection of synthetic images for semantic segmentation of urban scenes. 2016.