Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

b-it Bonn-Aachen International Center for Information Technology

R&D Project

Semantic Segmentation using Resource Efficient Deep Learning

Naresh Kumar Gurulingan

Submitted to Hochschule Bonn-Rhein-Sieg,
Department of Computer Science
in partial fullfilment of the requirements for the degree
of Master of Science in Autonomous Systems

Supervised by

Prof. Dr Paul G. Plger
M. Sc. Deebul Nair

August 2018

I, the undersigned below, declare that this work has not previously been submitted to this or any other university and that it is, unless otherwise stated, entirely my own work.

Date	Naresh Kumar Gurulingan

Abstract

Your abstract

Acknowledgements

Thanks to ....

Contents

1

Introduction

1.1 Motivation

1.1.1 ...

1.1.2 ...

1.2 Challenges and Difficulties

1.2.1 ...

1.2.2 ...

1.2.3 ...

1.3 Problem Statement

1.3.1 ...

1.3.2 ...

1.3.3 ...

2

State of the Art

2.1 ....

Use as many sections as you need in your related work to group content into logical groups

Don't forget to correctly cite your sources [4].

2.2 Limitations of previous work

3

DCNN and Semantic Segmentation

3.1 What is semantic segmentation

3.2 Traiditional methods

3.3 Deep Learning for Semantic Segmentation

3.4 Artificial Neural Networks

3.5 Convolutional Neural Networks

4

Dataset creation

4.1 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 4.1.

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.

4.2 Selection of a labeling tool

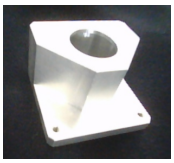
In order to reduce the time required to annotate an image, it was imperative to select a tool which is specifically designed for semantic segmentation and also provides algorithms which helps the annotator by providing labeling automation to the highest possible extent.

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

(a) axis [1]

(b) bearing [1]

(c) bearing box AX01 [1]

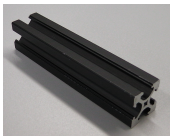


(d) bearing box AX16

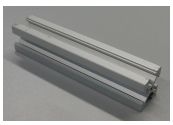
(e) container blue [1]

(f) container red [1]

(g) distance tube [1]



(h) F20_20_B [1]



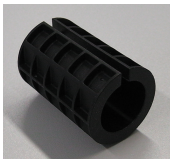
(i) F20_20_G [1]

(j) M20 [1]

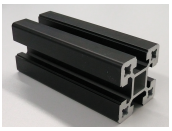
(k) M20_100 [1]

(l) M30 [1]

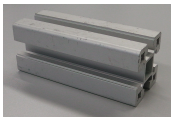
(m) motor [1]



(n) R20 [1]



(o) S40_40_B [1]



(p) S40_40_G [1]

(q) em_01

(r) em_02

Figure 4.1: Different objects required in the dataset

- 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).

4.3 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 figure4.2.

The ImageLabeler app, by default, provides different tools which help create pixelwise labels4.3. 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.
- 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.

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 4.2: Contents of the labelDefs .mat file

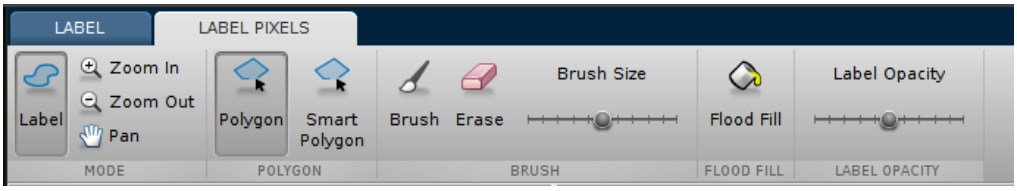


Figure 4.3: Tools provided by the ImageLabeler app

- 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 window4.4.

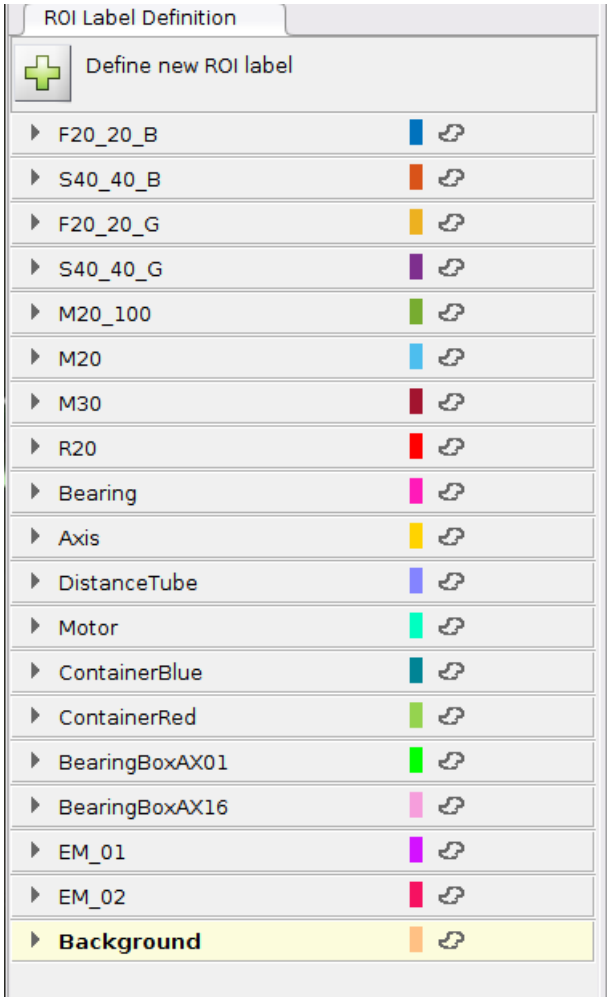


Figure 4.4: ROI Label Definitions window

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 4.5a.

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 4.5a) is the label image. This image is renamed to have the same name as the image file and a folder structure as in 4.5b is created by using a python code.

(a) Folder structure of saved labels

(b) Rearranged folder structure

Figure 4.5: Different folder structures

The final folder structure is shown in 4.6. The image folder and label folder are similar and contain object images and corresponding label images with same names.

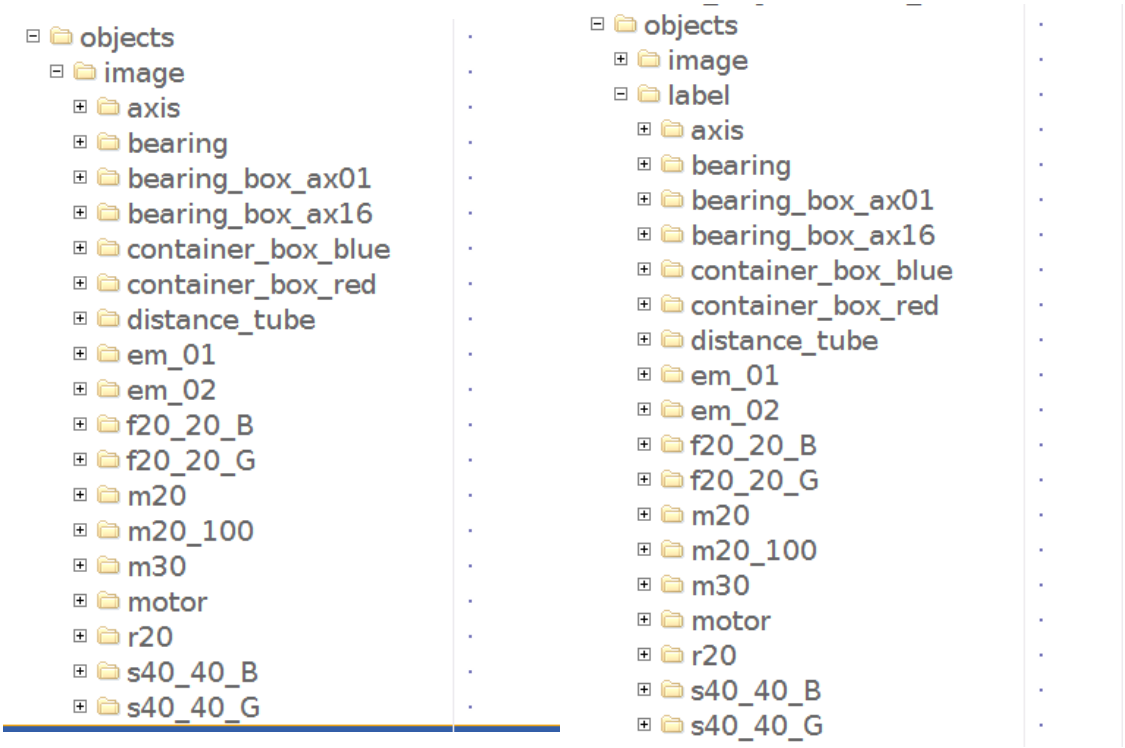


Figure 4.6: Folder structure showing different object folders in both image and label folders.

4.4 About the artificial image generation algorithm

4.4.1 Motivation

- Manually labeling 540 images with the described process in 4.3 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.

4.4.2 Working

The artificial image generation algorithm requires that every image provided must have just one object. The algorithm can be used in two modes named "Generate artificial images" and "Save visuals". The first mode can be used to generate artificial images and if required save visualizations of labels. The second mode can be used to just save visualizations. Under the first mode, the entire process can be divided into 7 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 **Visualize results**: The generated images and labels are visualized to verify the generation process.

7 **Save results**: The obtained resultant artificial images, corresponding labels and generated visualizations are saved.

Under the second mode, steps 3 to 6 in the above process is skipped. Step 2 fetches also the object detection labels in addition to the image and semantic labels. Step 7 saves the visualizations of read image, semantic labels and object detection labels.

4.4.3 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 4.1 and 4.2.

4.4.4 Sample results

Sample results of the artificial image generation algorithm can be seen in 4.7. The bounding box represents the object detection label and the different colors of the segmentation labels denote different label values. The colors were chosen in such a way that they are as distinct from each other as possible[3].

4.4.5 Downloading background images

Different background images were used for the artificial image generation process. Since a large number of backgrounds were required, manual download was time

Generator options	Description
mode	1: Generate artificial images; 2: Save visuals.
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.
obj_det_label_path	Path to directory where the object detection csv 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.
save_overlay	Save segmentation label overlaid on image.
overlay_opacity	Opacity of label on the overlaid image.
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.
overlay_save_path	Path where overlaid images 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 4.1: Description of generator options

Figure 4.7: 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.

consuming. Hence, the "google-images-download" [2] script was used to auto download images. The search keywords used to obtain the background images are

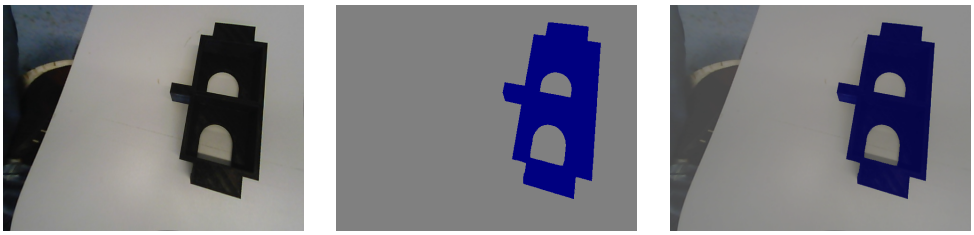
listed in 4.3. From the downloaded images, the rquired number of images for each dataset split were selected. For the shades of white dataset, many different search keywords were tried as is evident from the 4.3. This was because many of the downloaded images did not contain sufficient white regions. Images which were not of the dimensions used in the dataset (480×640) were rescaled.

4.4.6 Notable features of the artificial image generator

In this section, certain features of the artificial image generator which are noteworthy are listed.

- The generator automatically creates object detection labels in addition to semantic labels. The object detection labels are obtained by finding the rectangle points which describe a bounding rectangle around the semantic labels.

- The generated artificial images and labels can be visualized in three different ways. A preview which shows the image alongside the generated labels. A mask image showing the different classes in different colors. An overlay image in which the generated labels are overlayed on top of the corresponding generated images. The opacity of overlay can be configured through generator options. Examples of visualizations can be seen in 4.8 and 4.7



(a) Image containing object em_01

(b) Corresponding segmentation mask

(c) Corresponding overlay visualization

Figure 4.8: This figure shows examples of two different types of visualization, ”mask” and ”overlay”. The third type ”preview” can be seen in figure 4.7

4.4.7 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 4.5.2.

4.5 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.

4.5.1 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 4.9 and 4.10. 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 4.9: A similar viewpoint of bearing box ax01 and bearing box ax16 where the difference in shapes between the two objects is clearly visible.

(a) A viewpoint of bearing box ax01

(b) A viewpoint of bearing box ax16

Figure 4.10: A similar viewpoint of bearing box ax01 and bearing box ax16 where the appear similar in shape.

4.5.2 Dataset variants

Four different dataset variants have been created as listed below:

- **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 4.4.

- **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 4.5. 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 4.6.

- **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 4.7.

4.5.3 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 unlikely 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". Sample visualizations of artificial images generated for this dataset can be seen in 4.11.

4.6 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.

4.6.1 Surface area of the objects

In order to comprehend the reasons as to why an object constitutes a certain percentage of pixels in a dataset split, the surface area of the objects in the real world could be considered. It is natural to assume that the percentage of pixels occupied by the objects is roughly proportional to the surface area of the object. However, not all objects have a strictly defined geometric shape. For this reason, the closest geometric shape to each object is assumed to calculate the surface area. The table provides details about the assumed geometric shapes and the surface areas for each object.

4.6.2 Analysing the "atWork_full" variant:

The total pixels occupied by each class in each of the training, validation and test sets is calculated. Next, this count of pixels is converted to percentage with respect to the total number of pixels in the corresponding dataset split. Also, the number of images in which each class of objects appears is counted and called class count. The resulting plots are shown in 4.12 and 4.13. In 4.12, it is evident that compared to the background class, the object classes occupy almost negligible pixel area. This is desired as the background class is present in all images and in most of them, occupies the most pixel area. The larger object classes occupy the most

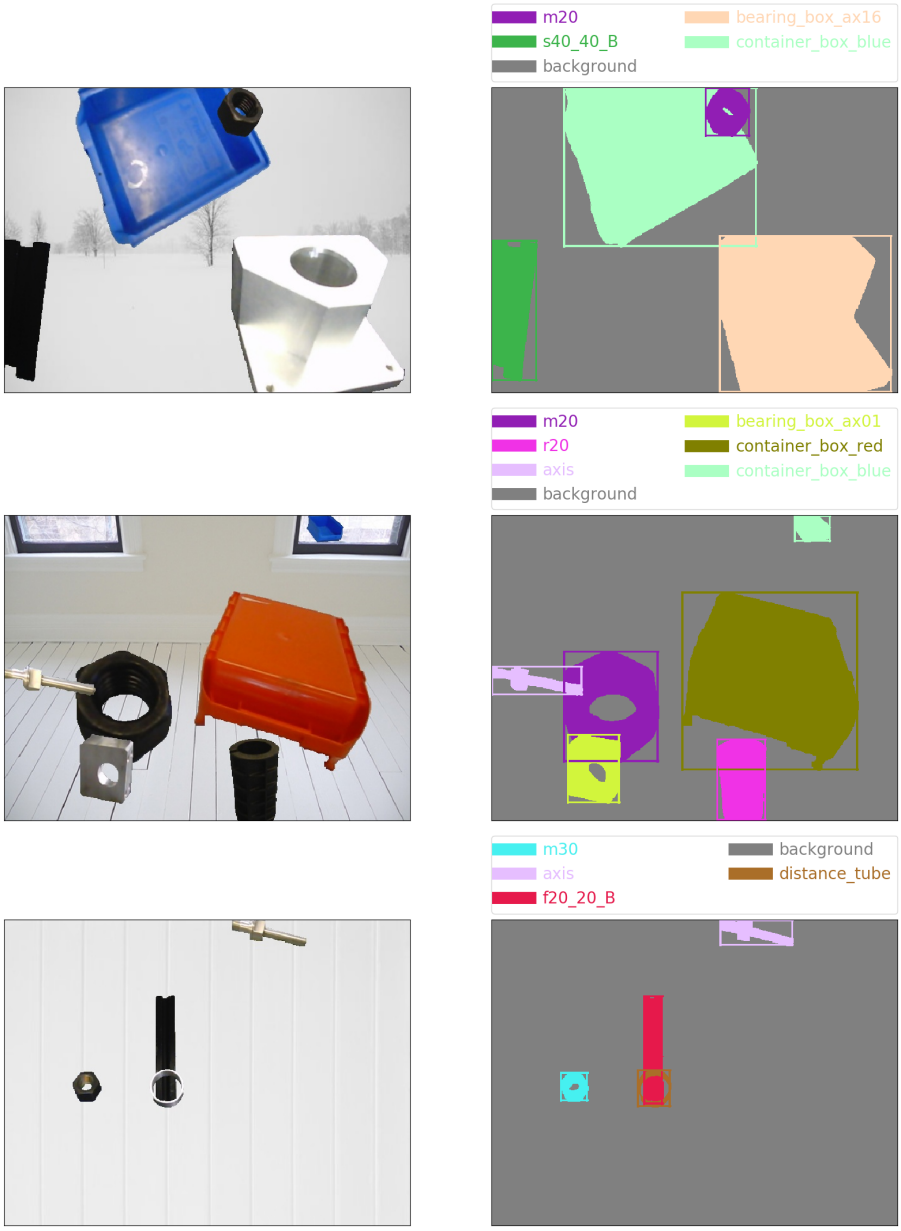


Figure 4.11: Sample results produced by the artificial image generation algorithm for the "shades of white" dataset.

pixel area as can be seen in 4.13. However, in terms of class count all objects are fairly close to each other. A direct reason is infact that larger size occupies more pixel area. Indirectly however, the larger objects are less likely to be rejected by the artificial image generation algorithm after being randomly scaled. This again

leads to increased pixel area occupied by the larger objects.

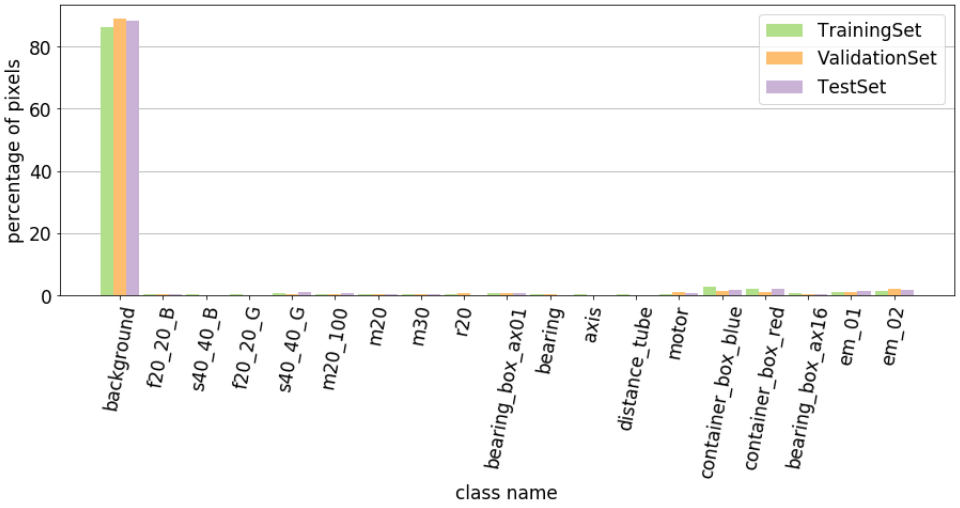


Figure 4.12: Percentage of pixels occupied by every class in the atWork_full variant. With respect to the background, the other classes occupy negligible number of pixels.

4.6.3 Analysing the "atWork_size_invariant" variant:

4.6.4 Analysing the "atWork_similar_shapes" variant:

4.6.5 Analysing the "atWork_binary" variant:

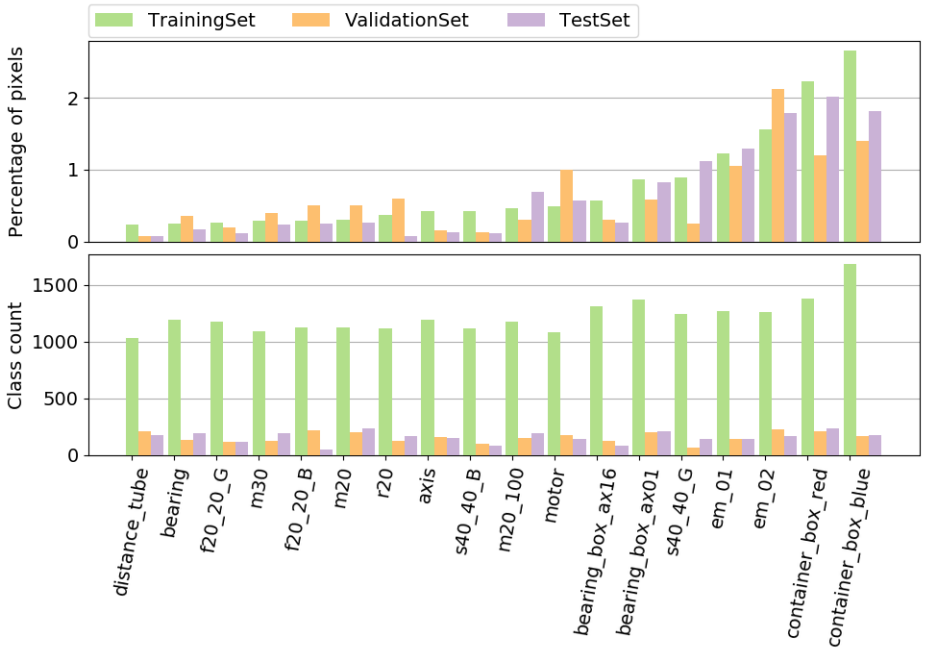


Figure 4.13: Percentage of pixels occupied by every class, except the background class, and corresponding class counts in the atWork_full variant. The larger objects such as "container_box_blue" occupy more number of pixels in comparision to smaller objects such as "distance_tube".

4.7 Meta-data of the dataset

Meta-data of the dataset is provided in table 4.9. These numbers hold true for all four dataset variants and also for the shades of white dataset. Initially, 30 images were captured for each of the 18 objects leading to a total of 540 images. However, 1 image of "axis" object and 2 images of "s40_40_B" were removed as they were blurred.

4.8 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

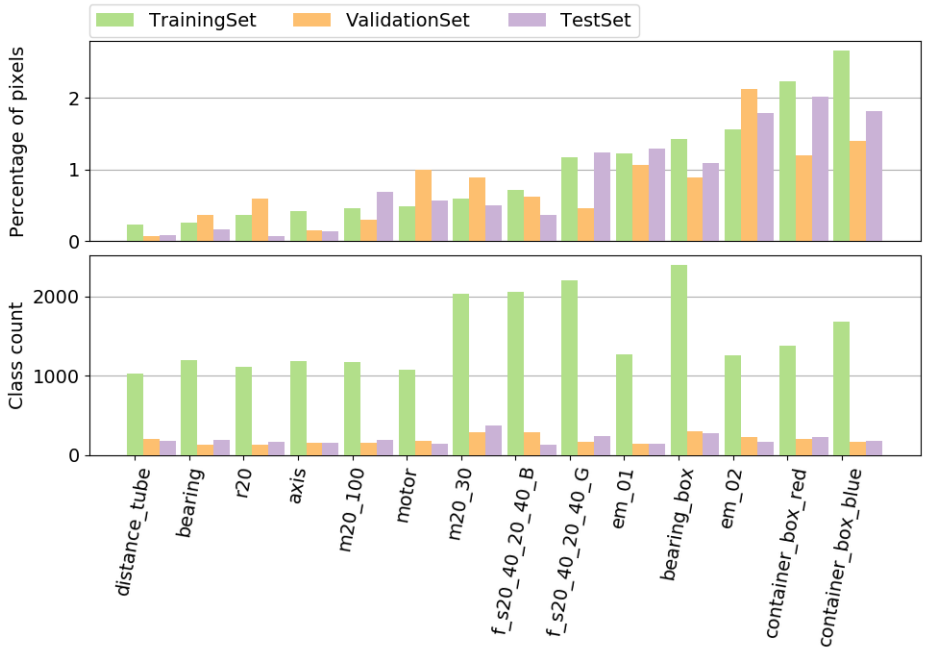


Figure 4.14: Percentage of pixels occupied by every class, except the background class, and corresponding class counts in the atWork_size_invariant variant.

method used in [5] and the Synthia dataset [6], takes a different approach. Unlike [5], 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 [6]. 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').

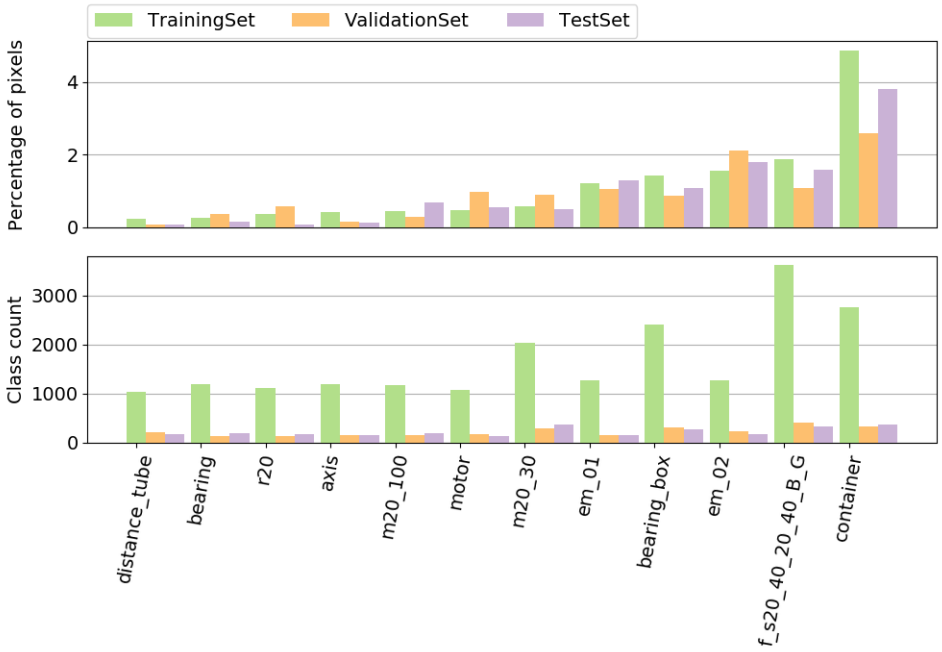


Figure 4.15: Percentage of pixels occupied by every class, except the background class, and corresponding class counts in the atWork_similar_shapes variant.

- 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.

Generator options	Default value	Is required?
mode	1	Not required
image_dimension	[480, 640]	Not required
num_scales	'randomize'	Not required
backgrounds_path	None	Required if mode is 1
image_path	-	Required
label_path	-	Required
obj_det_label_path	None	Required if save_label_preview is True and mode is 2
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
save_overlay	False	Not required
overlay_opacity	0.6	Not required
image_save_path	None	Required if mode is 1
label_save_path	None	Required if mode is 1
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
overlay_save_path	None	Required if save_overlay is True
start_index	0 if mode is 1 " if mode is 2	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 4.2: Default value of generator options and whether the options are required to be set.

Used in	Search keyword(s)	Number of images selected
Training set	640x480 background images, 640x480 textures images, 640x480 wallpapers	150
Validation set	640x480 abstract	25
Test set	640x480 paintings	25
Shades of white	640x480 white abstract, 640x480 white backgrounds, 640x480 white textures, 640x480 white wallpaper, light gray, white, white clouds, white floors, white frost, white mist, white pebbles, white snow, white table textures	150

Table 4.3: This table lists the keywords used to download images used as background for artificial image generation.

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 4.4: 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 4.5: Details of the "atWork_size_invariant" variant

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 4.6: Details of the "atWork_similar_shapes" variant

Variant Name	Object Name	Label Value
atWork_binary	background	0
	foreground	1

Table 4.7: Details of the "atWork_binary" variant

Object name	**Assumed shape**	**Surface area**
f20_20_B	cuboid	
s40_40_G	cuboid	
f20_20_G	cuboid	
s40_40_G	cuboid	
m20_100		
m20		
m30		
r20		
bearing_box_ax01		
bearing		
axis		
distance_tube		
motor		
container_box_blue		
container_box_red		
bearing_box_ax16		
em_01		
em_02		

Table 4.8: Assumed shape and surface area of objects

	Training	Validation	Test
Real Images	22 per object. Total: 22×18=396	4 per object. Total: 4×18=72	"axis"=3; "s40_40_B"=2; All other objects=4 Total: (4×18)-3=69
Augmented Images	7104	870	870
Total Images	7500	942	939

Table 4.9: Meta-data of all 4 variants and also the shades of white dataset.

5

Methodology

5.1 Semantic segmentation architectures

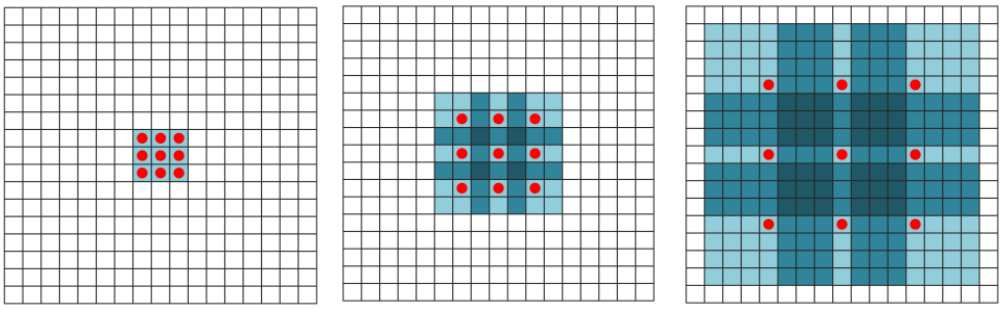
Inline with the goal of the project to use resource efficient deep learning in terms of inference time and storage memory, the deepLab v3+ model with mobileNetv2 and xception variant was chosen. In order to better understand deepLabv3+, we consider breaking down the architectures of the previous versions of deepLab leading upto the current version. The different versions of deepLab are deepLab, deepLabv2, deepLabv3 and deepLabv3+ (the current version also called as deepLabv4).

5.1.1 DeepLab

Fully Convolutional networks were introduced by [] for the task of semantic segmentation. The predictions obtained with the help of this network were coarse and the object boundaries were not sufficiently delineated. In order to overcome these difficulties, the authors of deepLab proposed the use of atrous convolutions and fully-connected Conditional Random Fields (CRF).

Atrous convolutions, also called as dilated convolutions, is be used to gather a better global context with enlarged field-of-view on the feature maps. A atrous rate called r, determines the field-of-view of an atrous convolution. With increase in dilation rate, a greater region in the feature map is convolved over. This leads to gathering of more global context. However, it is worth noting that there is no

increase in the number of parameters in the convolution filter. Only the convolved region in the input feature map changes. When the dilation rate is 1, usual convolution is performed. The figure 5.1 illustrates atrous convolution.



(a) Atrous rate = 1 (b) Atrous rate = 2 (c) Atrous rate = 3

Figure 5.1: Illustration of atrous convolution with three different atrous rates.

Fully-connected Conditional Random Fields (CRF), is used to post process the prediction of the CNN to improve object delineation. Every pixel in the output feature map is connected to every other pixel resulting in pairwise terms. In each pairwise term, based on color and position, the similarity between pixels is determined and a class is assigned for the pixels.

5.1.2 DeepLabv2

Source: http://www.davidtvs.com/exploring-semantic-segmentation-dnn/ In DeepLabv2, Atrous Spatial Pyramid Pooling (ASPP) was used in addition to the existing architecture. The authors also use deeper ResNet network to imrpove accuracy.

Atrous Spatial Pyramid Pooling, is used to create multiscale feature representations. Atrous convolutions with different atrous rates are applied to the same feature map. The resulting feature maps from each atrous convolution is processed in seperate branches in a similar fashion as in deepLabv1 by using two 1×1 convolutions. Each of the branches are then fused together to obtain multi-scale information. The difference between in architecture between deepLabv1 and deepLabv2 is illustrated in 5.3.

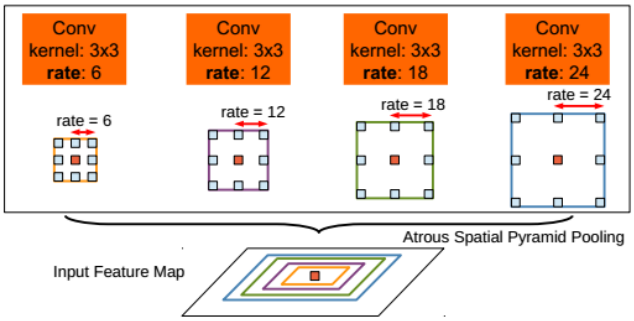
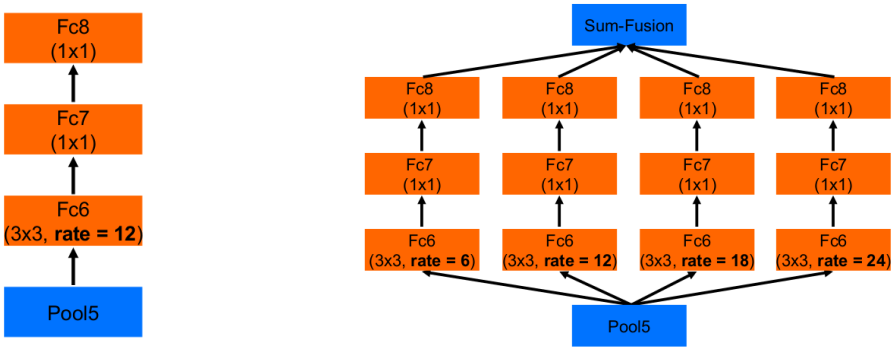


Figure 5.2: Illustration of Atrous Spatial Pyramid Pooling (ASPP). Atrous convolutions with 4 different rates convolve on the same input feature map. The field-of-view of each atrous rate is shown using different colors.



(a) Large field-of-view in DeepLabv1

(b) ASPP in DeepLabv2

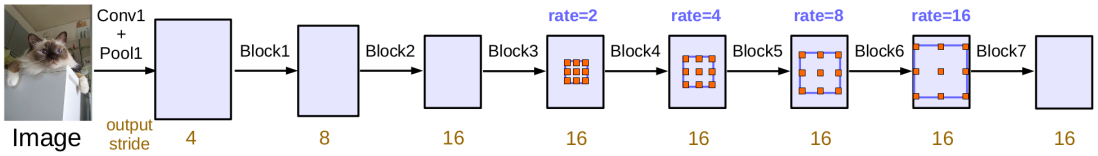
Figure 5.3: Illustration of the difference in architecture between DeepLabv1 and DeepLabv2.

5.1.3 DeepLabv3

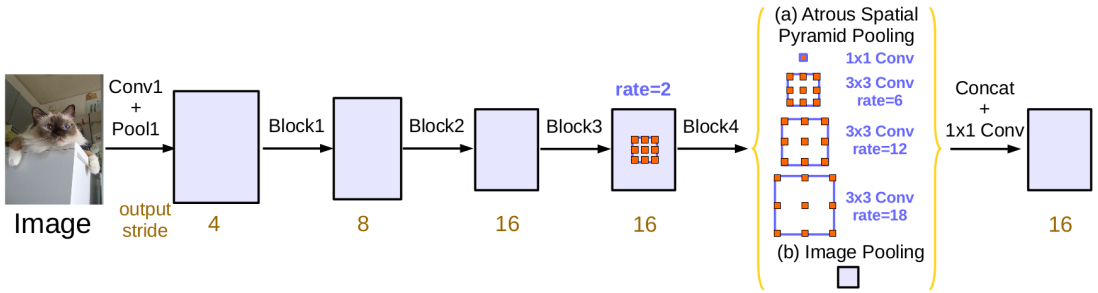
In this version of deepLab, the contributions include improvements to the context module, the use of batch normalization and multi-grid method.

The authors experiment with two different modules to handle context one being a cascade module and the other being an improved version of ASPP module. The cascade module is formed by repeating the last block from ResNet and replacing convolutions with atrous convolutions. The authors report that performing this repetition upto three times improves performance. The cascade module is illustrated

in **??**. The ASPP module used in deepLabv3 is similar to the one used in deepLabv2. However, there are a few changes including



(a) cascade module in DeepLabv3



(b) ASPP module in DeepLabv3

Figure 5.4: Illustration of two different context modules used in deepLabv3.

Batch Normalization is applied to every layer in the context module. This change leads to improvement in the Mean IOU attained in PASCAL VOC 2012 dataset.

Multi-grid method is employed to

5.1.4 DeepLabv3+

DeepLabv3+ is designed to combine the ability of ASPP module which can capture rich context information and the ability of encoder-decoder networks which can produce sharp object boundary delineation.

5.1.5 MobileNetv2

5.1.6 Xception

5.1.7 Quantization

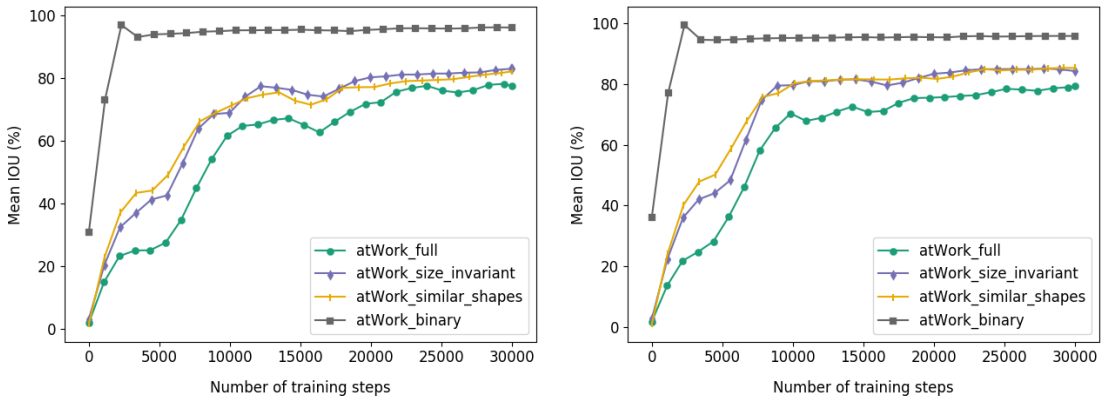
6

Experimental Evaluation

Since the major contribution of this work is the creation of the dataset, the experiments are focussed on validating the effectiveness of the dataset.

6.1 Comparing dataset variants

- **Objective**: The objective of this experiment is to compare the performance of deepLabv3+ on the different dataset variants.
- **Expected result**: The segmentation model is expected to perform better when the number of classes is lower. This is based on the notion that when similar objects are considered as different classes, the model would not have sufficient features to distinguish them.
- **Inference from the results**: Deeplabv3+ with both the mobileNet backbone and the xception backbone are evaluated on all variants of variety of backgrounds and white backgrounds dataset. From 6.1, it is evident that the Mean IOU obtained on each variant is dependent on the properties of objects in the variant. The atWork_full variant treats all the 18 objects in the dataset as different classes. As a result, for instance, m20 and m30 have different labels despite the fact that the two objects only differ in size and slightly in color. The segmentation model is thus forced to distinguish between such objects. Since the objects occur in the dataset in arbitrary scales and are subject to differences in illumination, the real world differences between such



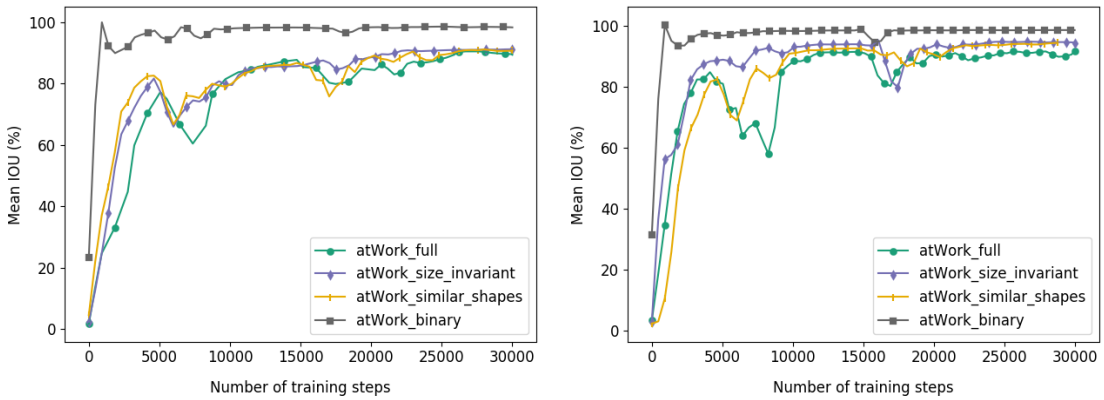
(a) (b)

Figure 6.1: Mean IOU of deeplabv3+ with **mobileNet backbone** on variety of backgrounds dataset and white backgrounds dataset is shown. (a) Mean IOU on the 4 variants of the variety of backgrounds dataset: atWork_full = 77.47%, atWork_size_invariant = 83.10%, atWork_similar_shapes = 82.10% and atWork_binary = 96.06%. (b). Mean IOU on the 4 variants of the white backgrounds dataset: atWork_full = 79.26%, atWork_size_invariant = 84.29%, atWork_similar_shapes = 85.33% and atWork_binary = 95.83%.

similar objects become insignificant in the dataset. Thus, the Mean IOU obtained in the atWork_full variant is indeed the lowest as expected. The two variants atWork_size_invariant and atWork_similar_shapes combine objects which are similar. As a result, the segmentation model achives better Mean IOU on these variants. The atWork_binary variant requries the segmentation model to only distinguish foreground from background leading to the highest MIOU. From 6.2, deepLabV3+ with the xception backbone, evidently, also follows a similar trend like deepLabv3+ with mobileNet backbone.

6.2 Comparing deepLabv3+ backbones

- **Objective**: The objective of this experiment is to compare the Mean IOUs obtained by deepLabv3+ with mobileNet and xception backbones on each of the datasets and its variants.
- **Expected result**: The xception backbone is expected to obtain higher Mean



(a) (b)

Figure 6.2: Mean IOU of deeplabv3+ with **xception backbone** on variety of backgrounds dataset and white backgrounds dataset is shown. (a). Mean IOU on the 4 variants of the variety of backgrounds dataset: atWork_full = 89.38%, atWork_size_invariant = 91.19%, atWork_similar_shapes = 90.81% and atWork_binary = 98.31%. (b). Mean IOU on the 4 variants of the white backgrounds dataset: atWork_full = 91.59%, atWork_size_invariant = 94.27%, atWork_similar_shapes = 94.33% and atWork_binary = 98.47%.

IOU because of the higher number of learnable parameters in comparison with the mobileNet backbone. In essence, the xception backbone has more "learning capacity" than the mobileNet backbone leading to the ability to learn a better decision boundary.

- **Inference from the results**: Across all the dataset variants, the xception backbone achieves higher Mean IOU than the mobileNet backbone consistently. Another inference is that the models trained and validated on the white backgrounds dataset perform slightly better than corresponding models trained on variety of backgrounds dataset. This supports the notion that it is easier to distinguish the atWork objects when the image background is predominantly white.

6.3 Training with different data

- **Objective**: The objective of this experiment is to assess the effectiveness of the created artificial data. On this regards, starting from the same initial weights, deepLabv3+ with each of the two network backbones is trained on:

  1 Entire training set of the variety of backgrounds dataset consisting of both real and artificial images.

  2 Only the artificial images in the training set of variety of backgrounds dataset.

  3 Entire training set of the white backgrounds dataset consisting of both real and artificial images.

  4 Only the real training images.

  The validation set only consists of the real validation images in order to consider only real world conditions.

- **Expected result**: Training with the entire training set of the variety of backgrounds dataset is expected to achieve the highest Mean IOU. This is based on the notion that the artificial images forces the segmentation model to learn features independent of the background. Also, the model

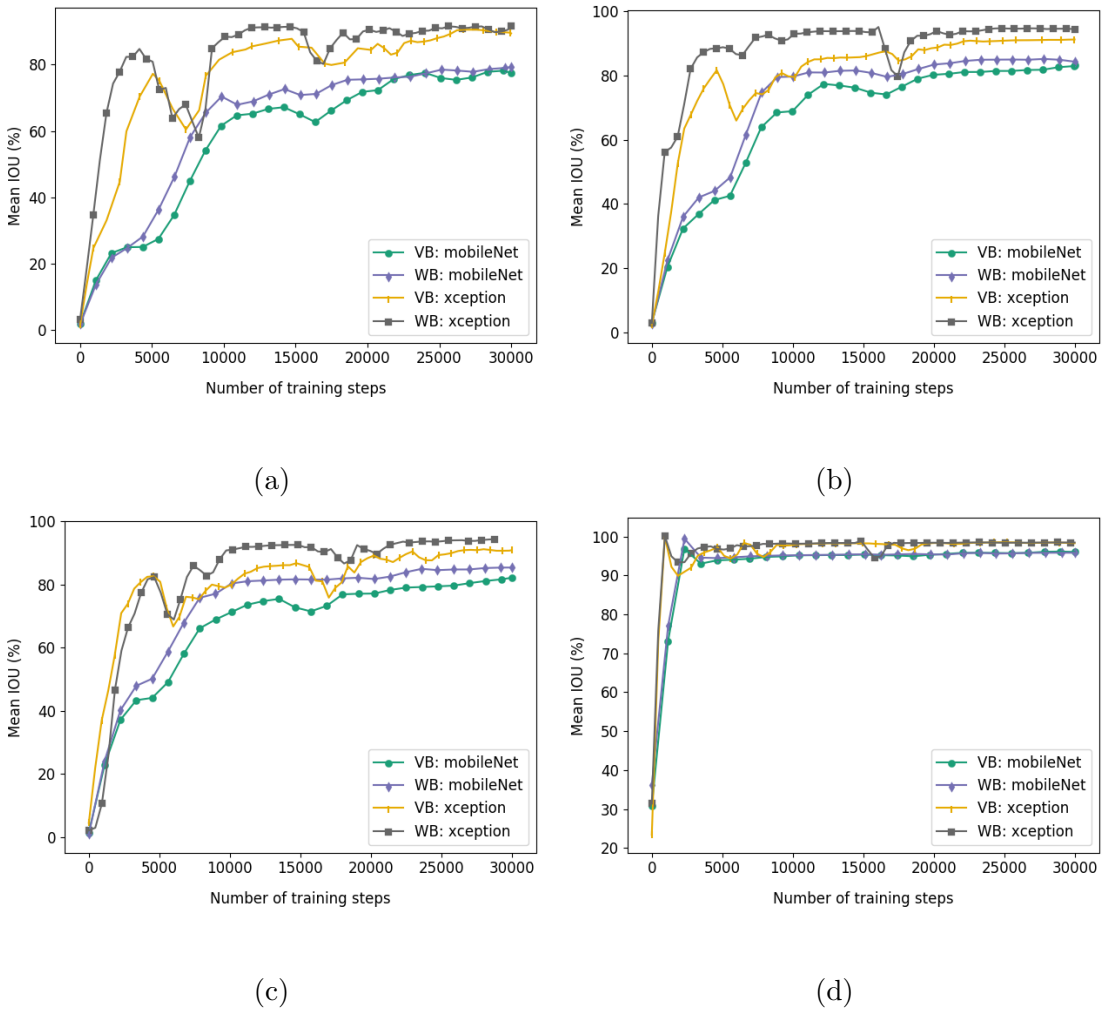


Figure 6.3: Comparision of Mean IOU obtained by deepLabv3+ using mobileNet backbone Vs xception backbone on all 4 variants. VB denotes variety of backgrounds dataset and WB denotes white backgrounds dataset. The dataset variant is (a): atWork_full variant, (b): atWork_size_invariant, (c): atWork_similar_shapes and (d): atWork_binary.

is expected to have improved robustness towards varying object scales and occlusions. Training with just the real training images is also expected to perform well but second to the performance obtained with the entire variety of backgrounds training set. Training with white backgrounds dataset is expected to perform well except in cases where the background is not predominantly white. Training only with the artificial images is expected to perform the worst has it does not introduce the segmentation model to real world conditions.

- **Inference from the results**: The results show that in all variants, training with just the real images achieves the best performance on the real validation set. This is in contrast to the notion that augmenting with artificial images improves performance. This apparant contrast begs to question the need for artificial images and states that they are not required. However, looking into the limitations of the real validation set could help reinstate the importance of the artificial images.

  1 The real validation images only contain one object per image which in most images is clearly visible. There is no cases of occlusion or existance of multiple objects.

  2 The backgrounds in the real validation set is already seen in the training set. Only three different real backgrounds were used.

  These two limitations exist in the real validation becuase of the need to reduce the labeling cost. Creating real world variations interms of multiple objects per image and random occlusions is time consuming and also leads to increase in annotation time. Introducing varied backgrounds in real images is also time consuming. These limitations are addressed by the artificial images by placing objects at arbitrary scales in random locations on varied backgrouds. In addition to the existing limitations, the artificial images inherently impose a regularization effect on the training process. This can be attributed to the existence of many different backgrounds. On this regard, the existing L2 regularization weight decay term might need to be lowered to enable the model to better fit to the training data.

Variant	Real training data	Variety of backgrounds all training data	White backgrounds all training data	Variety of backgrounds artificial training data
atWork_full	83.21	71.72	70.8	40.0
atWork_size_invariant	85.01	80.08	77.12	47.76
atWork_similar_shapes	79.83	77.33	76.47	43.31
atWork_binary	94.33	93.01	90.17	43.29

Table 6.1: This table summarizes the results obtained when validating only on the real validation data. The first column denotes the variant. The remaining columns denote on what data was the deepLabv3+ with mobileNet backbone model trained on. All the Mean IOUs are in percentage.

Variant	Real training data	Variety of backgrounds all training data	White backgrounds all training data	Variety of backgrounds artificial training data
atWork_full	87.03	80.26	78.42	45.67
atWork_size_invariant	90.84	89.58	87.67	41.58
atWork_similar_shapes	92.85	87.76	83.58	43.32
atWork_binary	98.19	95.21	94.31	47.91

Table 6.2: This table summarizes the results obtained when validating only on the real validation data. The first column denotes the variant. The remaining columns denote on what data was the deepLabv3+ with xception backbone model trained on. All the Mean IOUs are in percentage.

- Suggestions to improve the experiment: Adding a limited number of real validation images which have multiple objects and occlusions, reducing the value of L2 weight decay are two possible changes which can be introduced to arrive at a better inference. However, at this point in order to validate these speculations, model trained only on real data is validated on artifical data. [**result to be attached**] In this case, the Mean IOU obtained is only around 40 percent proving that this speculation could be further explored.

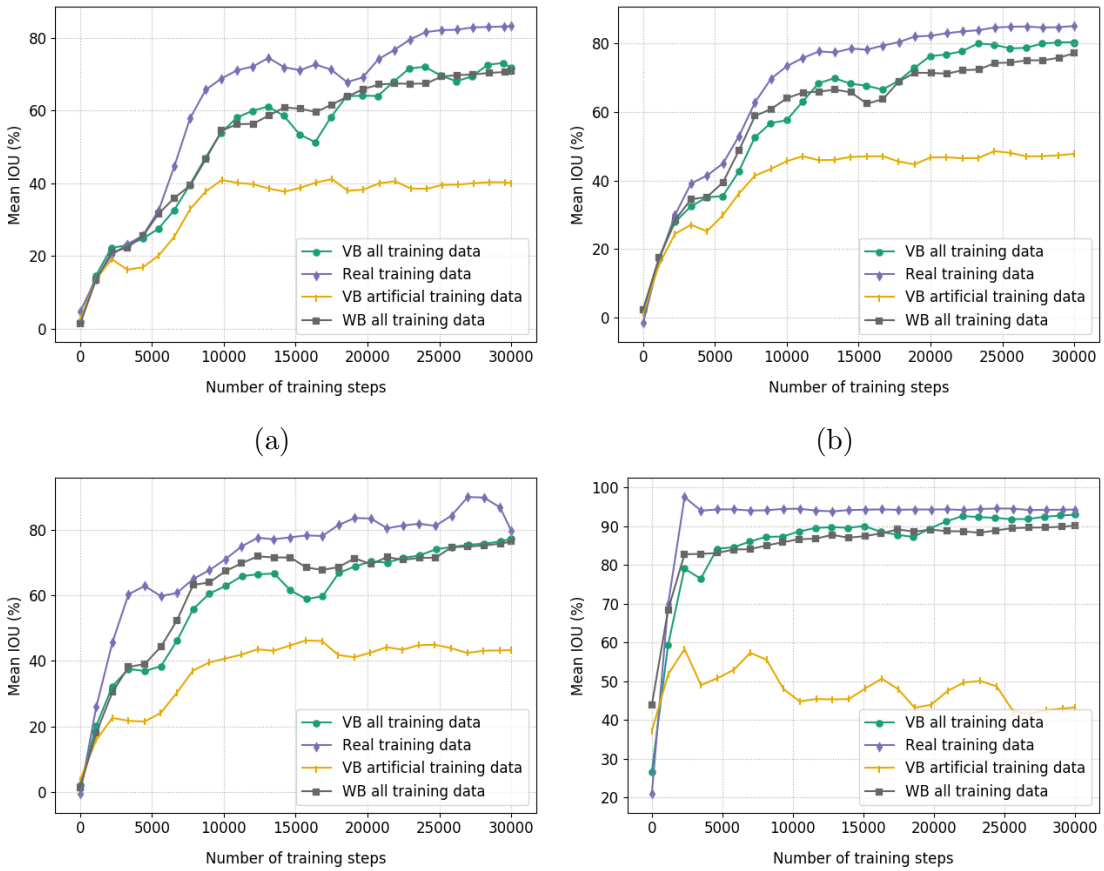


Figure 6.4: Mean IOU on all 4 variants obtained by deepLabv3+ with mobileNet backbone when validated only on the real validation data. VB stands for variety of backgrounds dataset and WB stands for white backgrounds dataset. (a): atWork_full variant, (b): atWork_size_invariant, (c): atWork_similar_shapes and (d): atWork_binary. The Mean IOUs are tabulated in **??**

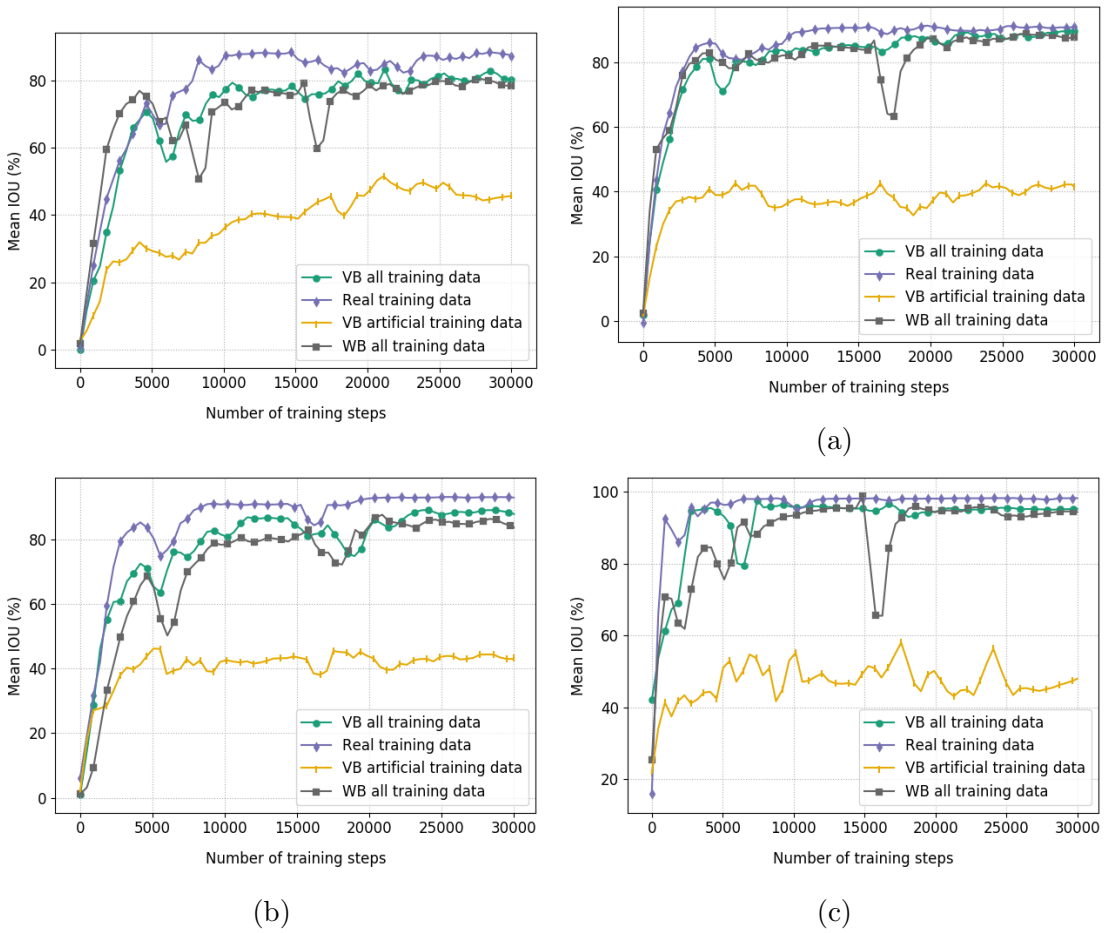


Figure 6.5: Mean IOU on all 4 variants obtained by deepLabv3+ with xception backbone when validated only on the real validation data. VB stands for variety of backgrounds dataset and WB stands for white backgrounds dataset. (a): atWork_full variant, (b): atWork_size_invariant, (c): atWork_similar_shapes and (d): atWork_binary. The Mean IOUs are tabulated in **??**

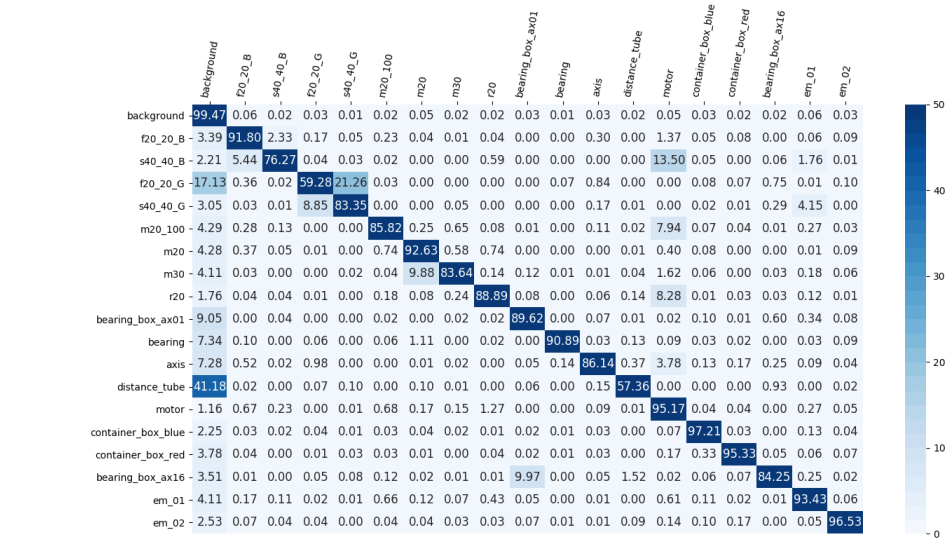
6.4 Comparing individual classes

6.4.1 Confusion matrix

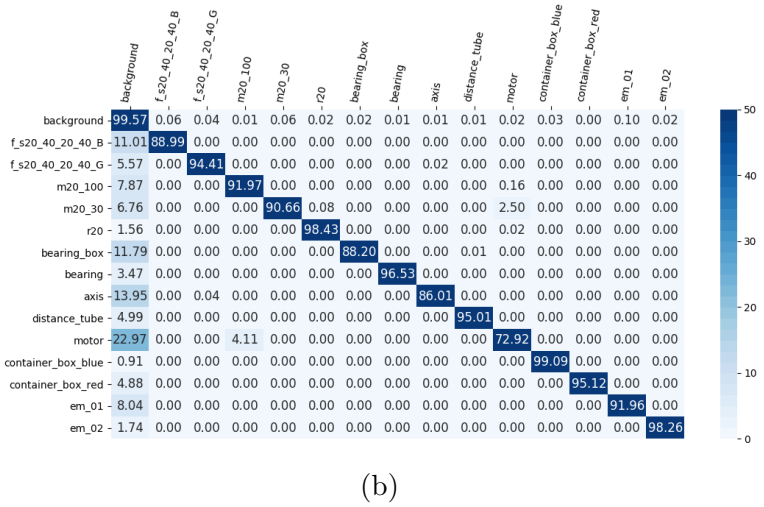
- **Objective**: The objective of this section is to analyze the deepLabv3+ models inability to distinguish between different objects.

- **Expected result**: The variants with higher number of classes is expected to have more non leading diagonal terms in the confusion matrix. This is based on the belief that the segmentation model would face difficulties distinguishing objects very similar to each other. This problem is expected to be alleviated by the atWork_size_invariant and atWork_similar_shapes variants. On the atWork_binary variant, there is a possibility that a certain percentage of foreground pixels are confused with background.

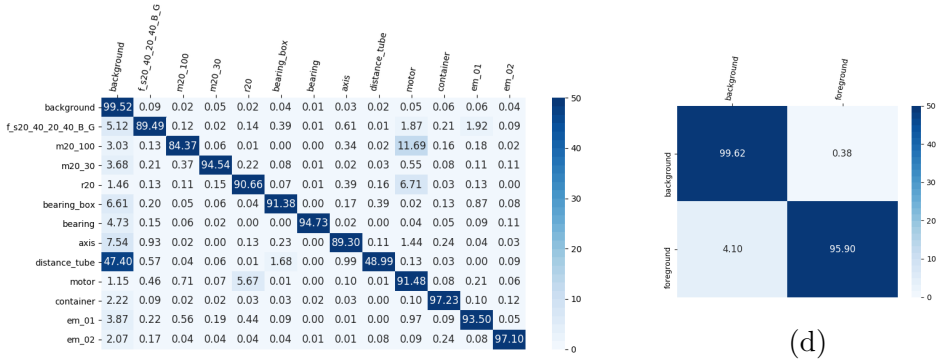
- **Inference from the results**: On all the confusion matrices, the leading diagonal elements have highest values in each row. This is expected and suggests that the model correclty classifies a majority of pixels in each class. Notably, the objects confused with each other are either similar in terms of color or shape. For instance, around 10 percent of m30 is confused as m20. This is reasonable as the objects are similar in shape and only differ in size. The difference in size cannot be picked up by the model as no consistent information regarding the object size is available in the dataset. 41.18 percent of pixels in distance tube are confused with background. This could be because the number of pixels occupied by distance tube in the dataset is small in comparision to the other objects. Confusions between objects has reduced on the atWork_size_invariant and atWork_similar_shapes dataset in comparison to the atWork_full variant. This can be attrbuted to the combining of similar objects to one class. The confusion between motor and m20_100 and the confusion between motor and r20 needs to be addressed.



(a)



(b)



(c)

(d)

Figure 6.6: Confusion matrix of deepLabv3+ with mobileNet backbone based on number of classified pixels on all 4 variants of the variety of backgrounds dataset. The number of pixels in each row is normalized by the total number of pixels in the row. (a): atWork_full variant, (b): atWork_size_invariant, (c): atWork_similar_shapes and (d): atWork_binary.

6.4.2 Class IOUs

- **Objective**: The objective of this experiment is to look for a relationship between the individial class IOUs and the percentage of pixels occupied by each class in the dataset.
- **Expected result**: With increase in percentage of pixels, the class IOU is expected to increase. This is based on the notion that the segmentation model gives preference to objects which dominate the dataset.
- **Inference from the results**:

6.5 Comparing learning rate decay policies

- **Objective**:
- **Expected result**:
- **Inference from the results**:

6.6 Effects of class balancing

- **Objective**:
- **Expected result**:
- **Inference from the results**:

6.7 Effects of quantizing the inference graph

- **Objective**:
- **Expected result**:
- **Inference from the results**:

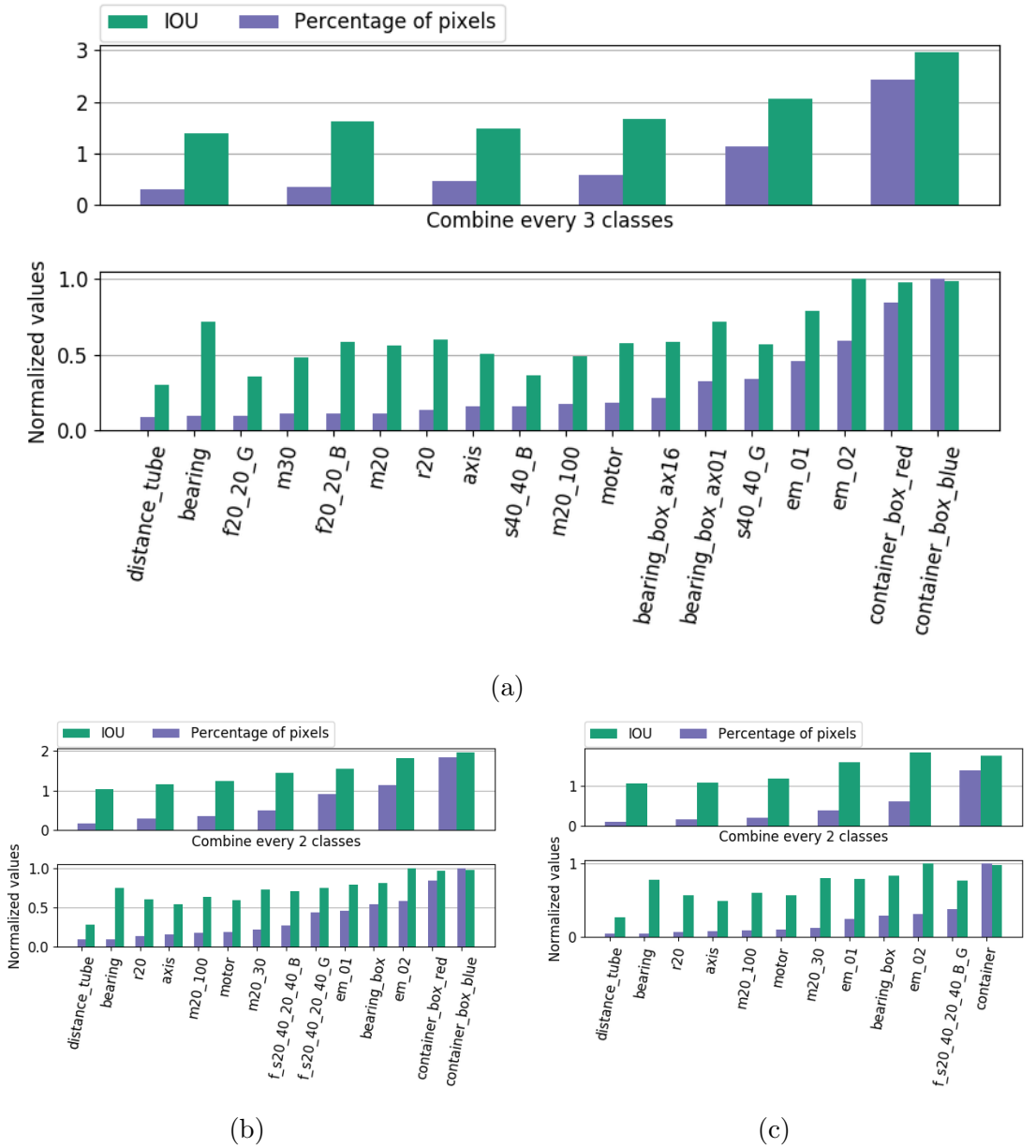


Figure 6.7: Individual class IOUs achieved by deepLabv3+ with mobileNet backbone is plotted with the percentage of pixels occupied on all 4 variants of the variety of backgrounds dataset. The number of pixels in each row is normalized by the total number of pixels in the row. (a): atWork_full variant, (b): atWork_size_invariant, (c): atWork_similar_shapes and (d): atWork_binary.

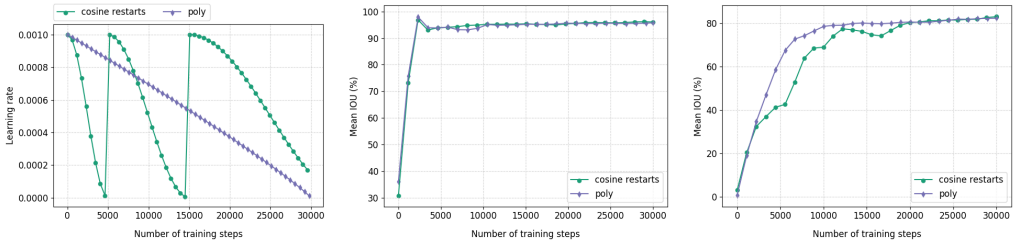


Figure 6.8: Learning rate decay with two different policies 1. cosine restarts and 2. poly is compared. Left: learning rate over 30000 steps with the two decay policies. Middle: Mean IOU on the validation set of atWork_binary variant is 96.06 % with cosine restarts and 95.75 % with poly. Right: Mean IOU on the validation set of atWork_size_invariant variant is 83.1 % with cosine restarts and 82.24 % with poly.

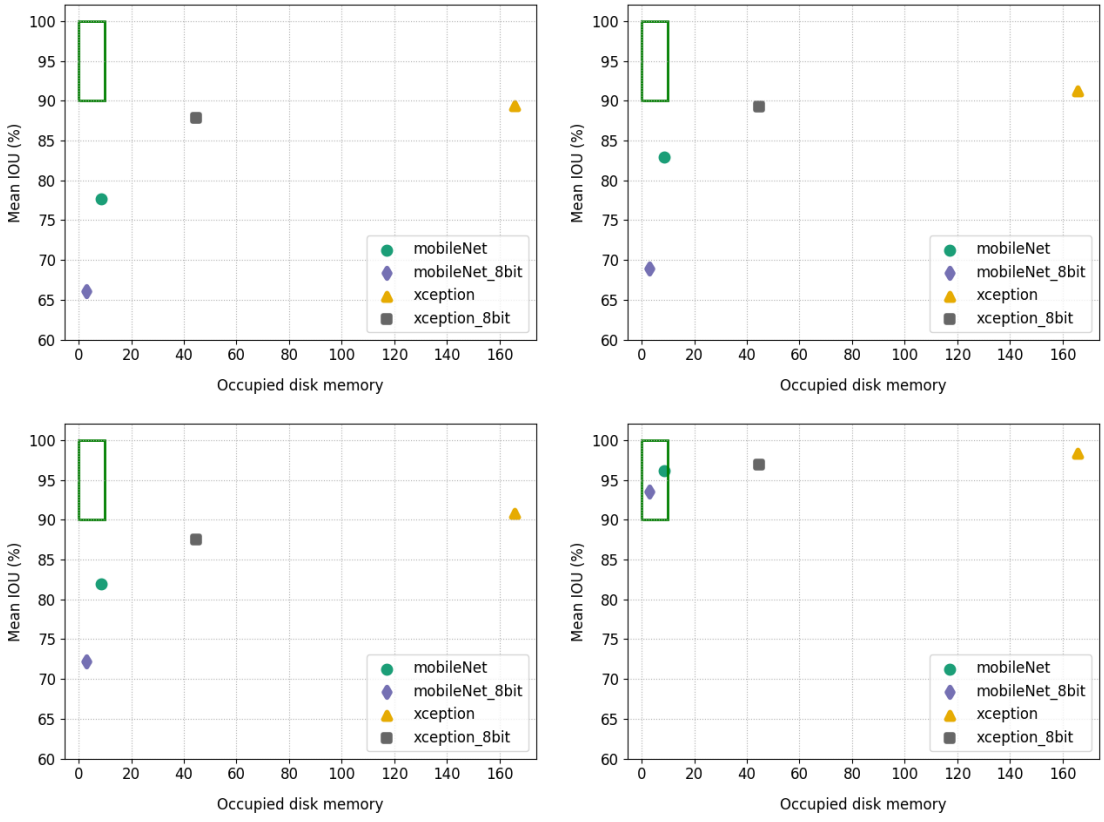


Figure 6.9: .

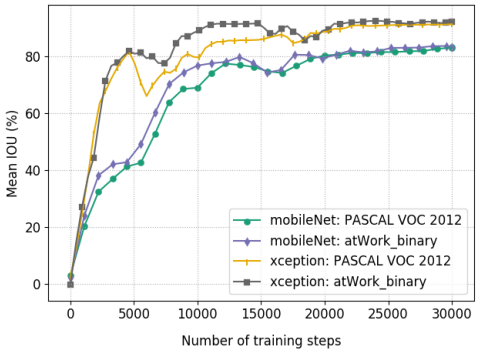


Figure 6.10: Size invariant;mobileNet: PASCAL VOC 2012 = 83.1, mobileNet: atWork_binary = 83.26, xception: PASCAL VOC 2012 = 91.19, xception: atWork_binary = 92.14

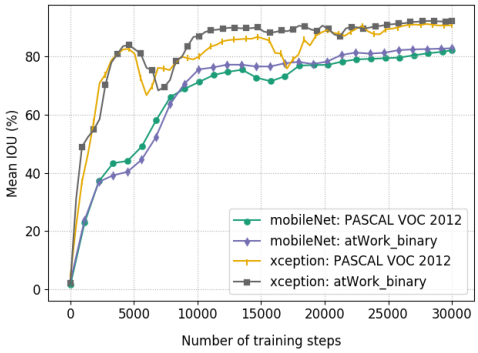


Figure 6.11: Similar shapes;mobileNet: PASCAL VOC 2012 = 82.1, mobileNet: atWork_binary = 82.8, xception: PASCAL VOC 2012 = 90.81, xception: atWork_binary = 92.15

6.8 Transfer learning

- **Objective**:
- **Expected result**:
- **Inference from the results**:

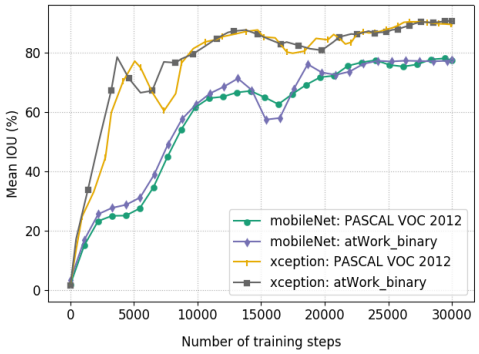


Figure 6.12: Full;mobileNet: PASCAL VOC 2012 = 77.47, mobileNet: atWork_binary = 77.73, xception: PASCAL VOC 2012 = 89.38, xception: atWork_binary = 90.64

7

Conclusions

7.1 Contributions

7.2 Lessons learned

7.3 Future work

A

Design Details

Your first appendix

B

Parameters

Your second chapter appendix

References

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

[2] githib: hardikvasa/google-images-download; link: https://github.com/hardikvasa/google-images-download.

[3] List of 20 simple, distinct colors; link: https://sashat.me/2017/01/11/list-of-20-simple-distinct-colors/. URL `https://sashat.me/2017/01/11/list-of-20-simple-distinct-colors/`.

[4] Author Name. Book title. *Lecture Notes in Autonomous System*, 1001:900–921, 2003. ISSN 0302-2345.

[5] Param S. Rajpura, Manik Goyal, Hristo Bojinov, and Ravi S. Hegde. Dataset augmentation with synthetic images improves semantic segmentation. *CoRR*, abs/1709.00849, 2017. URL `http://arxiv.org/abs/1709.00849`.

[6] 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.