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. Ploeger
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

In recent years, deep learning has significantly impacted research in the field of computer vision. Variations of Convolutional Neural Network architectures have shown state-of-the-art performance in computer vision tasks such as image classification [19], object detection [30], action recognition [35] and semantic segmentation [11]. A considerable part of this success comes from the supervised learning paradigm through which the networks are trained with labeled samples.

State-of-the-art deep learning techniques in semantic segmentation also make use of the supervised learning paradigm. Semantic segmentation is treated as a pixel-wise classification problem with the goal of assigning a class from a list of desired classes to every pixel in an image. The resultant image splits objects of interest into different regions thereby achieving the intended segmentation into meaningful regions.

Unlike the task of image classification (Figure 1.1a), where the location and boundaries of the desired object in the image are irrelevant, semantic segmentation (Figure 1.1b) requires the neural network to be able to spatially localize desired objects, delineate object boundaries and produce single channel segmentation map (output image) which is of the same size as the input image.

1.1 Motivation

Semantic segmentation provides rich information from image space which could be interpreted to make useful inferences about the real world scene depicted by

(a) (b)

Figure 1.1: (a) In image classification, we just get the output "em_01". (b) In semantic segmentation, we get an output image with each pixel labeled from which we can infer that the image consists of "em_01" and "background".

the image. We look at three potential applications of semantic segmentation which stand out to show the benefits of information obtained through semantic segmentation.

1.1.1 Potential applications

Semantic segmentation can be used in autonomous cars to segment a road scene and extract information such as where the road is, where pedestrians are and so on. Figure 1.2a shows an example of one such road scene segmentation. In robotics, an example application would be the segmentation of an indoor dining table shown in Figure 1.2b. A robot could use this information to identify plates, chairs and so on. In augmented reality, an augmented dog could be placed on a sidewalk to walk a person to his destination as illustrated in Figure 1.2c.

1.2 Challenges and Difficulties

In this section, we look at the difficulties posed by semantic segmentation in a deep learning setting.

(a) (b) (c)

Figure 1.2: (a) Street scene [14], (b) Indoor environment [34], (c) Augmented guide [26].

1.2.1 Labeling cost

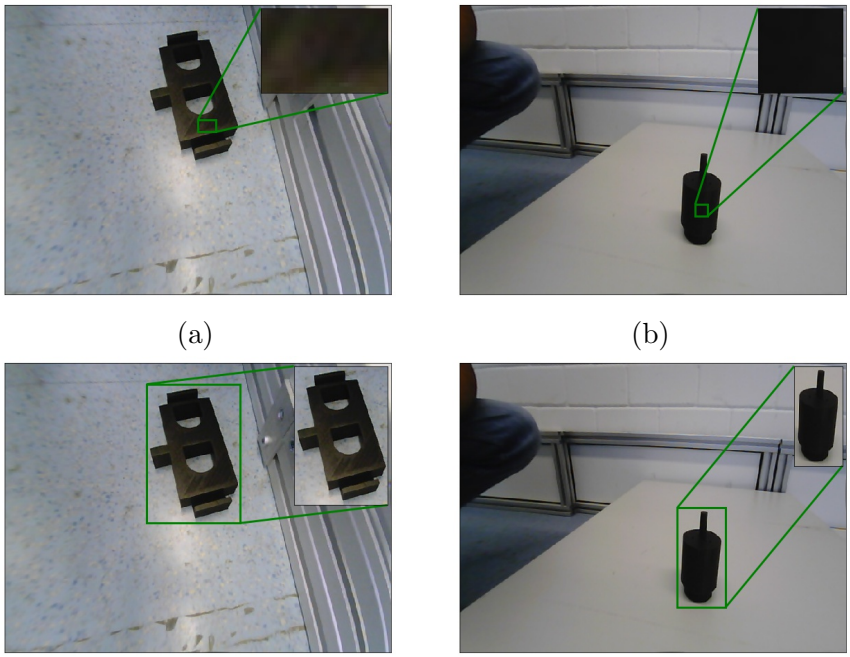
As semantic segmentation is a pixel-wise classification task, every pixel in the ground truth image needs to be labeled. This labeling, can be achieved by first performing accurate object boundary delineation and later annotate each region with the corresponding class. Since objects could have a variety of shapes, this boundary delineation becomes difficult. Object occlusions and image space cluttered with objects further add to this difficulty.

1.2.2 Context knowledge

Different features could represent objects in general. In a local context, where only a small region inside the object is considered, different objects might have very similar features. Figures 1.3a and Figure 1.3b illustrate the selection of two local regions within two different objects. By looking into just these selected regions, it is difficult to the classify the pixels within the regions as in a local context the two objects are similar. However, when a global context is gathered as illustrated in Figures 1.3c and 1.3d, the two objects become distinct and it is now possible to classify the pixels within these regions. Therefore, for the task of semantic segmentation, gathering global context is important.

1.3 Problem Statement

The problems intended to be addressed in this project are listed below:



(a) (b)

(c) (d)

Figure 1.3: Illustration of a local region in (a) "em_01" and (b) "motor". The two local regions provide local context. Illustration of region containing entire object shown in (c) "em_01" and (d) "motor". The regions illustrated in (c) and (d) can be said to provide global context of corresponding objects.

- Existing benchmark datasets are not suitable when the objects needed to be segmented do not exist in the benchmark dataset. Therefore, a semantic segmentation dataset needs to be created for Robocup @Work.

- Creating semantic segmentation ground truth images is time-consuming because of the need to delineate boundaries of @Work objects which have a variety of shapes.

- Since creating annotations is expensive, the dataset needs to be augmented with artificial images. The artificial images need to add diversity in terms of object scales, occlusions and so on.

- One or more semantic segmentation models need to be selected which are resource efficient in terms of both memory and inference time.

- The selected segmentation models are required to be trained to segment @Work objects with considerable accuracy and at the same time is required to be resource efficient.

2

State of the Art

In the field of deep learning for computer vision, two distinct lines of research could be observed. One line of research includes the likes of ResNet [19], and Inception [36] which push convolutional neural networks (CNN) deeper and wider. The focus here is to improve the accuracy on the task at hand. Another line of research is more concerned with the practical use of deep learning in embedded or mobile devices. Here, approaches tend to focus on finding the best hyperparameters which require the least resources but yet perform with considerable accuracy. The range of approaches include attempts to design a resource efficient architecture by changing the number of layers and so on, attempts to compress CNNs using techniques like pruning [24] and quantization [40], and attempts to automatically search through the hyperparameter space to find the best hyperparameters by using reinforcement learning [42] and evolutionary algorithms [29].

In this chapter, we review networks which focus on improving accuracy (Section 2.1) and networks which focus on both accuracy and resource efficiency (Section 2.2) on the task of semantic segmentation. We also look into compression techniques called pruning and quantization in Section 2.3.

2.1 Improving accuracy

In this section, we look into semantic segmentation networks which focus on improving accuracy.

2.1.1 Fully Convolutional Networks

Fully Convolutional Networks [22] extend the success of convolutional neural networks on image classification, to the dense prediction task of semantic segmentation. Fully connected layers of image classification networks are converted to convolutional layers, which enables the prediction of a segmentation map. Convolution and pooling layers downsample the input image to produce feature maps which have a low spatial resolution. These feature maps are then upsampled using transposed convolution layers which use fractional stride to increase in spatial resolution. Skip connections, as illustrated in Figure 2.1 are used to combine coarse high layer information with fine low layer information to help refine the predicted segmentation map.

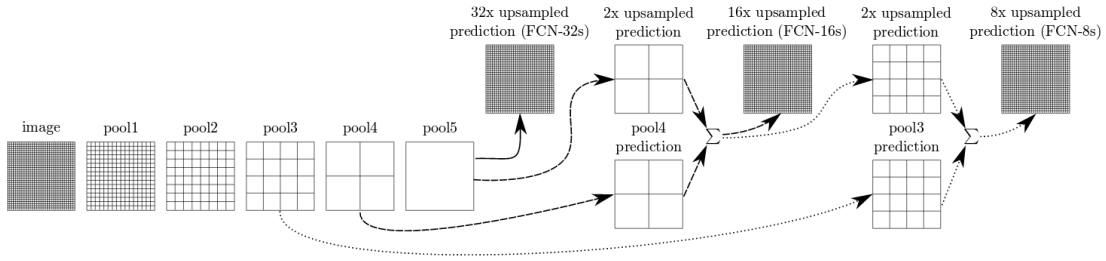


Figure 2.1: This figure illustrates the use of skip connections in Fully Convolutional Networks. Pooling layers downsample the input image. The pool5 layer results in 32 times reduction in spatial resolution of the input image. Three different predictions are shown. In the first, pool5 output is upsampled by 32. In the second, pool5 output is upsampled by 2, added with pool4 output with a skip connection and then upsampled by 16. In the third, pool5 output is upsampled by 2 two times, added with pool3 output using skip connection and then upsampled by 8. This third prediction is experimentally shown to produce finer predictions [22].

2.2 Accuracy and resource efficiency

In this section, we look into semantic segmentation networks which consider both accuracy and resource efficiency in terms of memory and inference time. These approaches focus on reducing resource requirements and also improve accuracy.

2.2.1 SegNet

SegNet [5] proposed the use of an encoder-decoder architecture for semantic segmentation as shown in Figure 2.2. In the encoder, since all the fully connected layers in the underlying VGG16 network is dropped, the number of parameters reduces from 134 million to 14.7 million. To downsample the input image, the authors use max pooling layers. The indices of values selected during max pooing are saved. In the decoder, the max pooling indices from the corresponding max pooling layer in the encoder are used to upsample the image. The authors state that this way of using max pooling indices in the decoder improves object boundary delineation and can be adapted in any encoder-decoder architecture. After upsampling, convolutional layers with trainable parameters are used in the decoder to refine predictions.

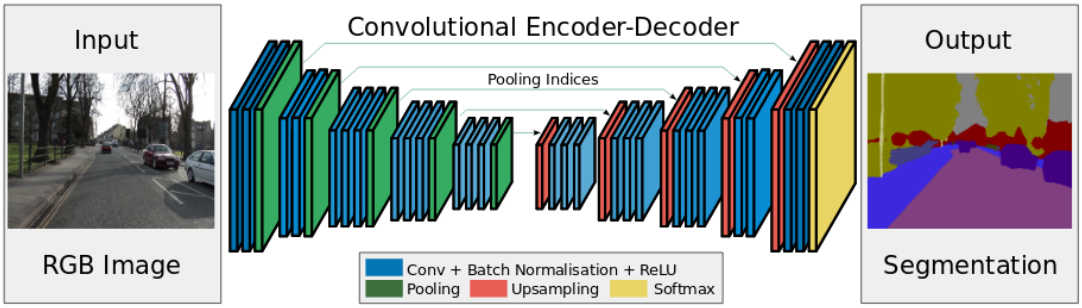


Figure 2.2: This figure illustrates the SegNet architecture. The decoder uses max pooling indices from corresponding encoder max pooling layers for upsampling [5].

2.2.2 ENet

ENet [25] is built with the goal of achieving fast inference with high accuracy. ENet also uses an encoder-decoder architecture similar to SegNet. The authors state that the initial layers of the encoder perform expensive computations as they operate on feature maps with high spatial resolution. Optimizing the initial layers will lead to improvements in inference speed. On this regard, ENet encoder heavily downsamples the input feature maps at the intial layers. The ENet initial

block shown in Figure 2.3, which performs early downsampling, downsamples the input image of resolution 512×512 and 3 channels to 16 feature maps of resolution 128×128. This block performs 3×3 convolution with stride 2 and max pooling with non-overlapping 2×2 windows, on the input image whose results are then concatenated. This early downsampling illustrates ENets approach to optimizing initial layers. In addition, unlike SegNet where the decoder has the same number of layers as the encoder, ENet uses a large encoder and a small decoder. The reduced decoder size is based on the notion that the decoder is only required to fine-tune features already gathered by the encoder.

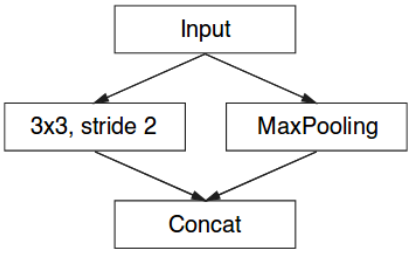


Figure 2.3: The ENet initial block which performs convolution with stride 2 and max pooling with 2×2 non-overlapping windows, on the input image to downsample the input image to half the original spatial resolution [25].

2.2.3 ICNet

ICNet [41] is designed with the intention of obtaining real-time semantic segmentation on high-resolution images. ICNet consists of a cascade of three branches operating on the input image at the original resolution, half of the original resolution and quater of the original resolution respectively. The predictions of the three branches are fused together by a cascade feature fusion unit to obtain fine predictions. Each of the branches is guided during training with the corresponding resolution of ground truth.

The number of layers in the three branches are 50, 17 and 3 respectively where the first branch operates on the lowest input resolution, the second branch operates on half the input resolution, and the third branch operates on the highest input resolution. The three branches result in feature maps which are 1/32, 1/16 and

1/8 times of the original image resolution respectively. A cascade feature fusion unit combines the feature maps of the three branches by using upsampling, dilated convolutions, batch normalization, and sum fusion. This architecture is illustrated in Figure 2.4. Overall, by using limited layers on highest resolution input and more layers as input resolution is lowered, computations is reduced when compared to a network with a single branch directly operating on the highest resolution.

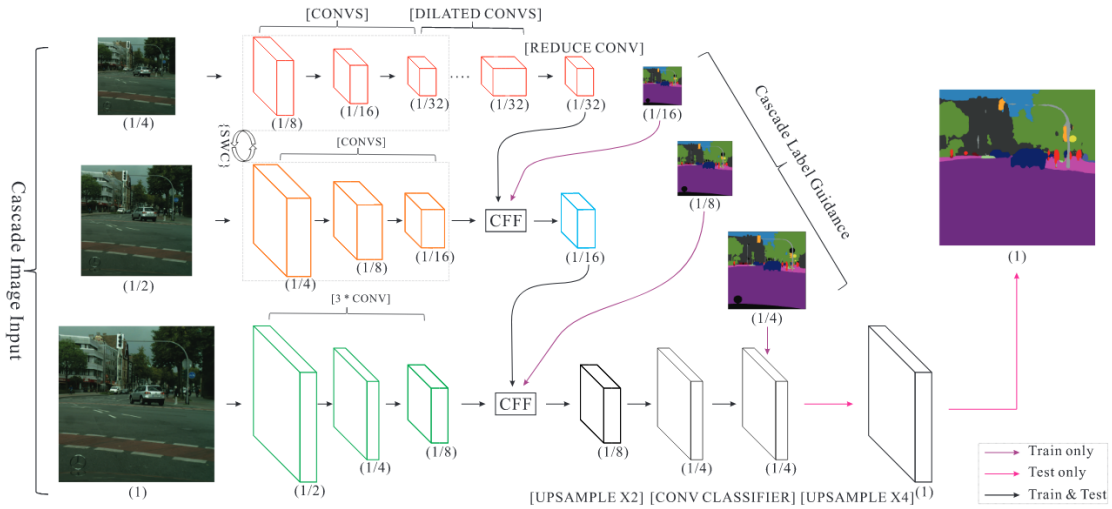


Figure 2.4: Illustration of the image cascade network (ICNet). The number in brackets refers to the ratio of the feature maps resolution to input resolution. CFF refers to the Cascade Feature Fusion unit. SWC refers to sharing weights between first and the second branch. Cascade label guidance is done only during training and upsampling to the original image resolution is done only during testing [41].

2.3 Compressing DCNNs

In this section, we look into approaches which compress Deep Convolutional Neural Networks (DCNN) by reducing the number of parameters or by using low precision calculations. The goal is to reduce resource requirements while retaining or incurring an acceptable loss in accuracy.

2.3.1 Pruning CNNs

Pruning is a technique through which redundant weights in a Deep Neural Network are removed. Weights are considered redundant if they do not contribute

significantly to the accuracy of the DNN on the task at hand.

In [24], the authors state that the prediction runtime is dominated by convolutional layers and removing entire feature maps will lead to speed up of inference. The CNN to be pruned is fine-tuned on the target task until convergence on the target task, after which iterations of pruning and fine tuning are alternated. Pruning is stopped once appropriate trade-off between accuracy and pruning objective such as minimum accuracy required or maximum allowed inference time, is met. The pruning procedure is illustrated in Figure 2.5. As a pruning criterion, an approximation of the change in loss function when a trainable parameter is removed by using first-order Taylor expansion is used. After pruning, a pruning gate is used to determine whether or not a feature map is included during inference.

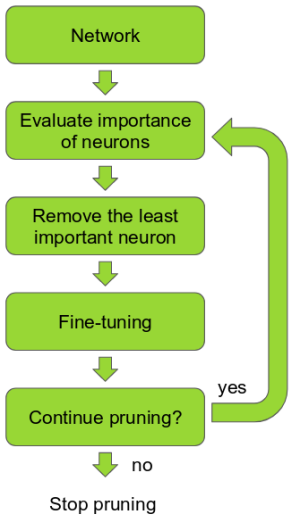


Figure 2.5: This figure illustrates the iterative procedure used for pruning [24].

2.3.2 Quantizing CNNs

Quantization is a technique through which the weights and calculations of trained neural networks are replaced with low precision equivalents in order to reduce memory and computation time.

In [40], the authors state that the convolutional layers take the most computation time and the fully-connected layers have the most amount of parameters. To

quantize fully-connected layers, the input space to every fully-connected layer is evenly split into M subspaces. For each of the subspace, a sub-codebook is learned using the sub-vectors within the subspace and the inner product between the sub-vectors and every sub-codeword in the sub-codebook is precomputed and stored in a look-up table. This process of quantizing fully-connected layers is illustrated in Figure 2.6. This look-up table leads to both reductions in computation time and storage. For convolutional layers, subspaces are created by splitting the filters along the dimension of feature maps. In this case, the precomputed look-up table is created for the inner product between feature maps and sub-vectors.

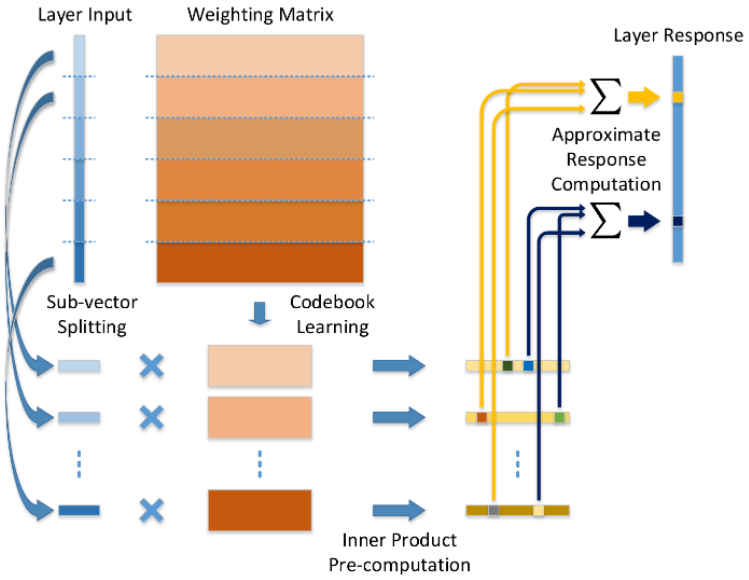


Figure 2.6: This figure illustrates the process of quantization of fully-connected layers. The input to a fully-connected layer is split into sub-vectors. A codebook with codewords is learned for the weighting matrix. The inner product between the sub-vectors and the codewords of the codebook are precomputed which can be used to obtain an approximate layer response [40].

3

Convolutional Neural Networks and Semantic Segmentation

In this chapter, we look into the basic concepts of neural networks and later, how such concepts have been used for the task of semantic segmentation. In section 3.1 we present an overview of Artificial Neural Networks. In 3.2 we look into basic concepts underlying Convolutional Neural Networks (CNNs). In 3.3 we look into how CNNs are adapted for the task of semantic segmentation.

3.1 Artificial Neural Networks

Artificial Neural Networks (ANN), inspired by the neural networks in our brain, was designed to learn tasks without explicitly programming descriptive features of the concerned tasks. An ANN is made up of processing units called neurons which performs a non-linear transformation, using an activation function, of the weighted linear combination of inputs. Figure 3.1a illustrates a non-linear model of a neuron. Many such neurons are connected to one another in an ANN resulting in its ability to learn highly non-linear function mappings from input to output space. Figure 3.1b illustrates a multilayer feedforward neural network which is a type of ANN.

3.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs or Conv Nets) are a class of feedforward neural networks which are used in the field of computer vision. Tasks such as

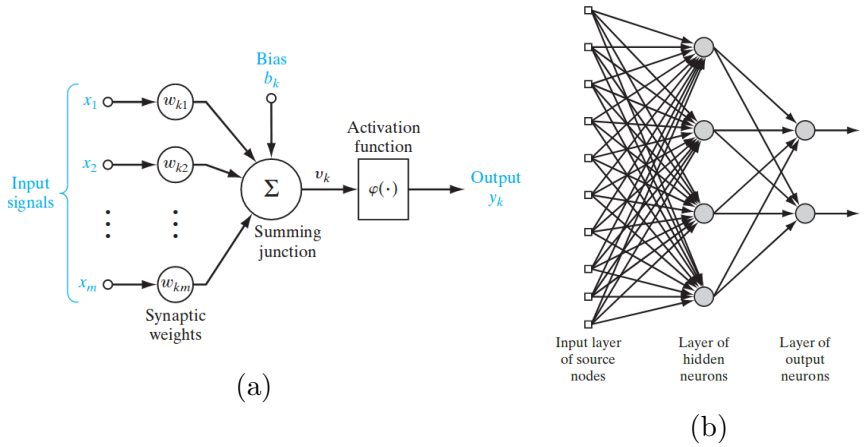


Figure 3.1: (a) A non-linear neuron model. $v_k = \sum_{i=1}^{m} w_{ki}x_i$ denotes the local field of the neuron, φ is the activation function and $y = \varphi(v_k)$ is the output of the neuron. (b) An illustration of a multilayer feedforward neural network with input source nodes, one hidden layer of neurons and an output layer of neurons [18].

image classification, object detection and semantic segmentation make use of CNNs. Unlike the multilayer feedforward network shown in Figure 3.1b, CNNs are designed to handle image data which is usually represented as a stack of 2 dimensional values. A major part of CNNs comprise the use of convolutional layers which greatly reduce the number of parameters in comparison to fully-connected layers.

3.2.1 CNN Architecture

The general architecture of a CNN is illustrated in Figure 3.2. A CNN is composed of different layer types such as convolutional layer, pooling layer and fully-connected layer. Each of the layers are looked into in detail in the subsequent sections.

Convolutional layer

An RGB image typically consists of height, width and 3 channels representing red, blue and green. In order to better handle this structure of an input image, the neurons in a convolutional layer are arranged in a 3D volume of height, width and depth. Each neuron is only connected to a small region in the input. This is called

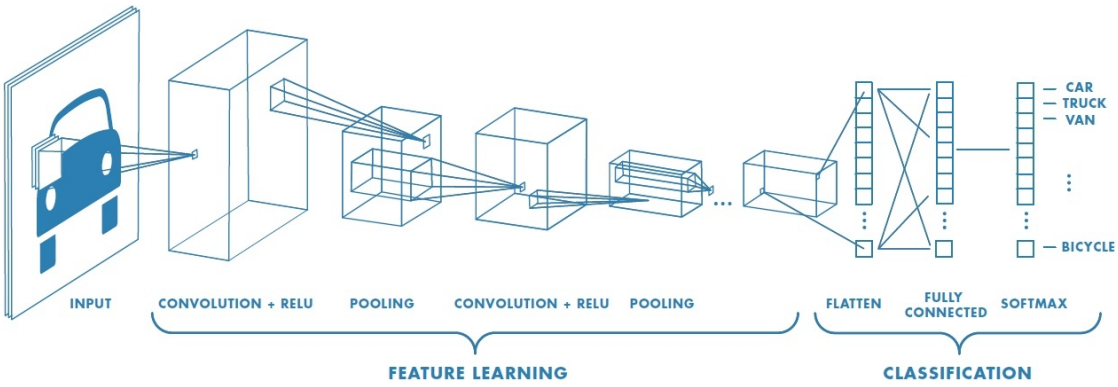


Figure 3.2: This figure illustrates a CNN architecture. A convolution layer convolves on the input image or feature maps followed by an ReLU activation which produces output feature maps. A pooling layer downsamples feature maps. The CNN architecture shown, performs feature learning and classification [4].

"local connectivity" and the extent of the input region to which a neuron connects is controlled by a hyperparameter called receptive field [3]. An illustration of the 3D arrangement of neurons can be seen in Figure 3.3. When moving along the depth dimension, each 2D arrangement of values are called filters or kernels.

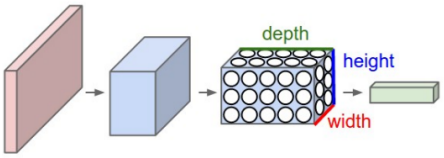


Figure 3.3: Illustration of 3D (height, width, depth) arrangement of neurons in a convolutional layer. The convolutional layer transforms a 3D input volume to a 3D output volume [3].

A filter slides through an input channel, performs convolution and produces output feature maps. A filter looks for specific features like edges and corners, at the input spatial location it convolves over. As an extention, a filter can be seen as a template image and it looks for the template across the input space. Figure 3.4 illustrates one such filter which looks for vertical lines.

The entire 3D convolution filter slides along the height and width of all channels of the input space looking for features along spatial locations and features across input channels. One 3D convolution filter results in a 2D feature map. Multiple

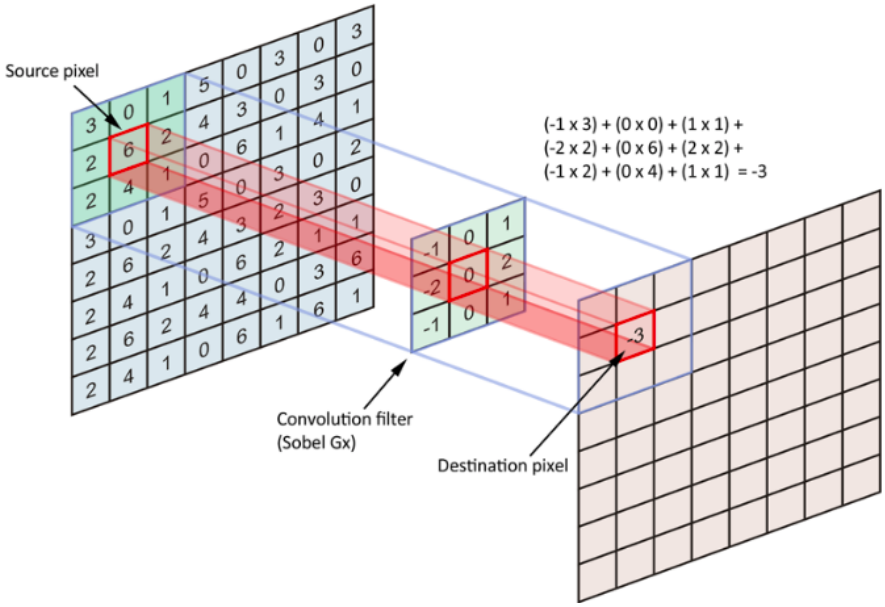


Figure 3.4: A Sobel kernel convolving over the input. This kernel looks for vertical edges in the input image. The kernel values are multiplied elementwise with the input locations and the results are added together to get the value at the location in the output feature map which corresponds to the center of the image locations [13].

such 3D convolutions are used each of which leads to a 2D feature maps which are all stacked along the depth dimension to create the output 3D volume. This process is illustrated in Figure 3.5.

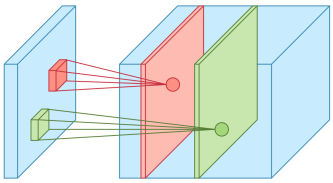


Figure 3.5: Illustration of 2D feature maps produced by two 3D convolution filters which are arranged along the depth dimension [15].

In CNNs, the filters are learnt through training the CNN for a specific task such as image classification. Once a CNN is trained, the filters in the lower layers often tend to learn simple features such as edges and corners and filters in the higher layers learn more abstract features building upon the features learned by filters in

the previous layer.

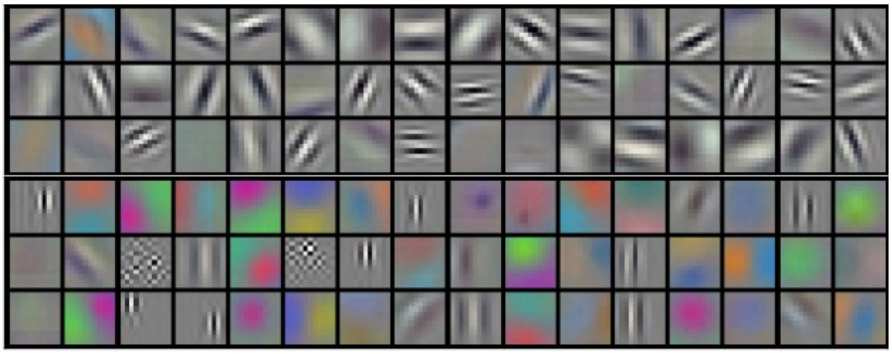


Figure 3.6: Learned filters of AlexNet CNN architecture. The filters have learned to detect features such as lines, blobs and so on [3].

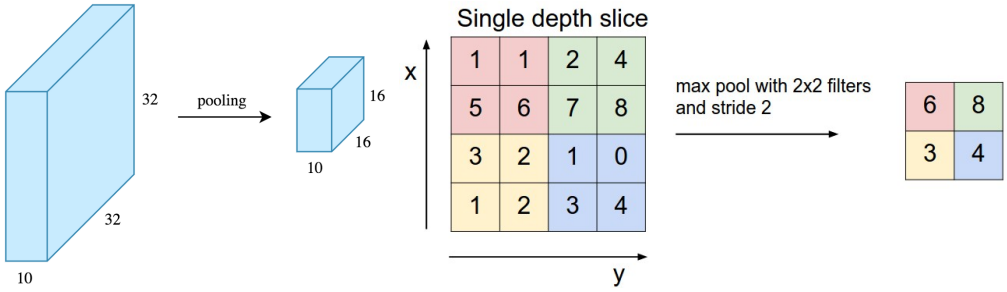
Pooling layer

A pooling layer is reponsible for reducing the spatial size of the feature maps leading to reduction in the number of trainable parameters in the network. This reduction in trainable parameters leads to reduced chances of overfitting and reduced computational cost. Additionally, pooling makes CNNs invariant to translations in the input image. This induced translation invariance is desired for tasks such as image classification where it is only required to find objects in an input space irrespective of where the object is located within the image.

A pooling filter looks at a particular window in a depth slice (one 2D feature map out of 3D input feature maps) of input feature maps and produces one value as output for the window. Two common pooling types are max pooling and average pooling. The pooling filter outputs the maximum of all values and average of all values for max pooling and average pooling respectively. An illustration of max pooling and reduction in spatial size of feature maps is shown in Figure 3.7.

Fully-connected layer

In fully-connected layers, every neuron in a layer is connected to all neurons or input nodes in the previous layer. The hidden layer shown in Figure 3.1b illustrates a fully-connected layer. In CNNs, the outputs produced by convolutional layers



(a) (b)

Figure 3.7: (a) Illustration of feature maps being downsampled by pooling [15]. (b) Illustration of max pooling where the pooling filter looks at 2×2 input windows. Stride 2 denotes that the window moves two steps when it moves through x or y. Each color in the depth slice of input feature maps indicates a different input window and the corresponding color in the output feature map shows the selected max value [3].

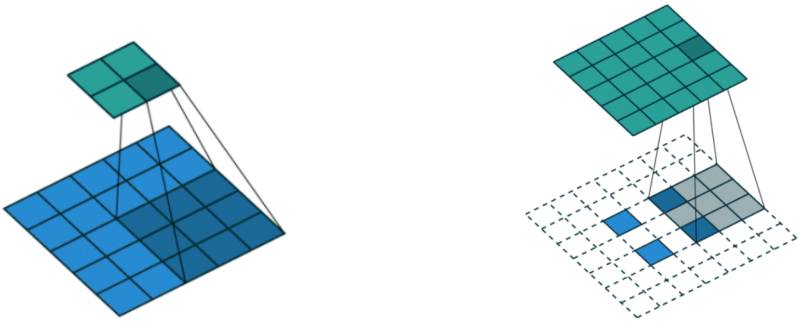
and pooling layers are 3D feature maps wheraes a fully-connected layer expects a 1D vector as input. Therefore, 3D feature maps are are converted to a 1D vector which is then given as input to fully-connected layers. There are two ways in which this 1D input to fully-connected layers is created in CNNs. One way is to directly flatten the 3D feature maps. For instance, if the 3D feature maps have a dimension of 7×7×5, the values are arranged in the form of a 1D vector with 245 values. In the second way, a technique called global average pooling is used to take the average value out of every depth slice of the 3D feature maps and then create a flattened 1D vector. If the feature map is of dimension 7×7×5, average value out of 5, 7×7 depth slice is taken to get 1×1×5. This is flattened to create a 1D vector input with 5 values as input to fully-connected layers. The output of fully-connected layers are then fed to a softmax function which converts the output into probability scores for the different classes involved.

3.3 CNNs for Semantic Segmentation

The CNN architecture discussed so far, learns to extract features and perform image classification by resulting in a probability score for each class. For instance, for a 10 class classification problem, for every input image, a trained CNN produces 10 values in the range [0,1] each of which provides the probability that the image

belongs to the corresponding class. However, semantic segmentation is a dense prediction task which requires a class to be assigned to every pixel in the input image and hence for every input image, the output is expected to be another image of the same spatial size as the input image and contains in every pixel a value which denotes a class. This output is called a segmentation map.

For this reason, two changes are done to the CNN architecture. One is replacing fully-connected layers with convolutional layers and the other is upsampling the final feature maps produced back to the original input spatial size. Fully-connected layers require a flattened 1D input vector and outputs a 1D vector. This is not required for semantic segmentation as the output is a 2D segmentation map. Hence, fully-connected layers can be replaced with convolutional layers. This has the advantage of reducing the number of trainable parameters as convolutional layers use less trainable parameters in comparison to fully-connected layers, leading to a reduction in the chances of overfitting.



(a) (b)

Figure 3.8: (a) 3×3 convolution with stride 2 on a 5×5 feature map (shown in blue) producing a 2×2 feature map (shown in green). (b) 3×3 fractionally strided convolution upsampling a 2×2 feature map (shown in blue) to get a 5×5 feature map (shown in green) [27].

After replacing the fully-connected layers with convolutional layers, we are left with 3D feature maps with reduced spatial size. To get a segmentation map, these 3D feature maps need to be upsampled back to the original input spatial size. One way to do this is to use an interpolation method such as bilinear interpolation to increase the spatial resolution. This is considered as a naive method of upsampling. An alternative would be to use fractionally strided convolutions (also known as

transpose convolution or deconvolution). This type of convolution as shown in Figure 3.8b, pads zeros to the input feature map in a manner which results in the desired upsampling when convolved over. In this example, we see a 2×2 feature map upsampled to 5×5 feature map by adding a row and a column of zeros in between the two existing rows and columns respectively and also zero padding by 2 around the feature map. A 3×3 convolution can now be used to get back a spatial size of 5×5. The advantage of using a fractionally strided convolution is that the parameters of the filter are learned through training instead of using a fixed upsampling method such as interpolation.

4

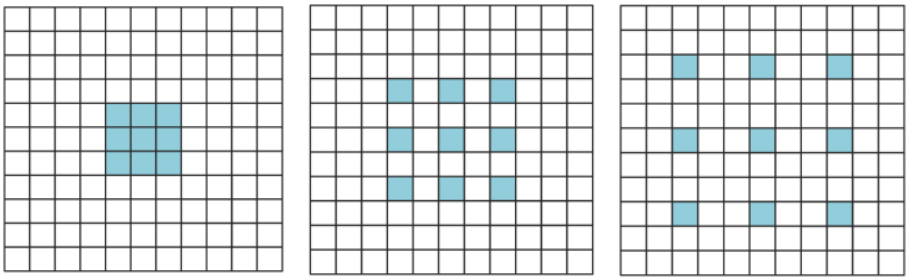
Methodology

In line 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 [33] and Xception [12] network backbones 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 [8], DeepLabv2 [9], DeepLabv3 [10] and DeepLabv3+ [11]. Also, we review the quantization method considered for this work.

4.1 DeepLab

Fully Convolutional Networks [21] were introduced by Long et al. 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 used to gather a better global context with enlarged field-of-view on the feature maps. An atrous rate "r", determines the field-of-view of an atrous convolution. With increase in atrous rate, a greater region of a 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



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

Figure 4.1: Illustration of 3×3 atrous convolution with three different atrous rates [17]. The blue squares denote the parameters of the convolution filter. The white grid space represents the input feature map. The field-of-view grows with increase in atrous rate but not the number of parameters.

region in the input feature map changes. When the atrous rate is 1, standard convolution is performed. The Figure 4.1 illustrates atrous convolution.

Fully-connected Conditional Random Fields (CRF), is used to post process the prediction of the segmentation network used in DeepLab, 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.

4.2 DeepLabv2

In DeepLabv2, Atrous Spatial Pyramid Pooling (ASPP) was used in addition to the existing architecture. The authors also use deeper ResNet network to improve 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 separate branches in a similar fashion as in DeepLabv1 by using two 1×1 convolutions. Each of the branches are then fused together to obtain multiscale information. The ASPP module is illustrated in Figure 4.2. The difference between the ASPP modules of DeepLabv1 and DeepLabv2 is illustrated in Figure 4.3.

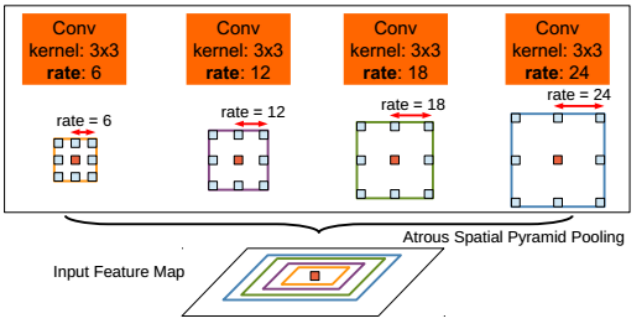
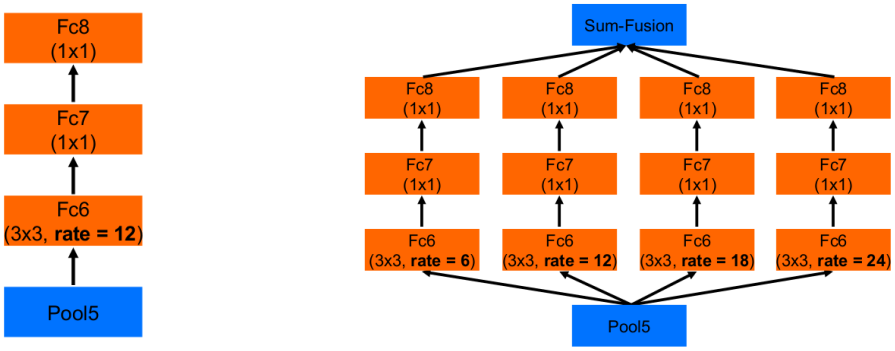


Figure 4.2: Illustration of Atrous Spatial Pyramid Pooling (ASPP). Atrous convolutions with 4 different rates convolve on the same input feature map. The field-of-view of each atrous rate is shown using different colors [9].



(a) Large field-of-view in DeepLabv1

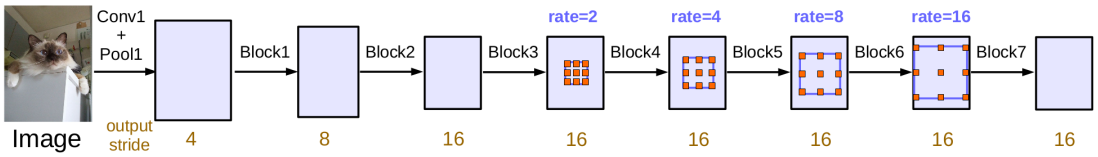
(b) ASPP in DeepLabv2

Figure 4.3: Illustration of the difference in architecture between DeepLabv1 and DeepLabv2 [9].

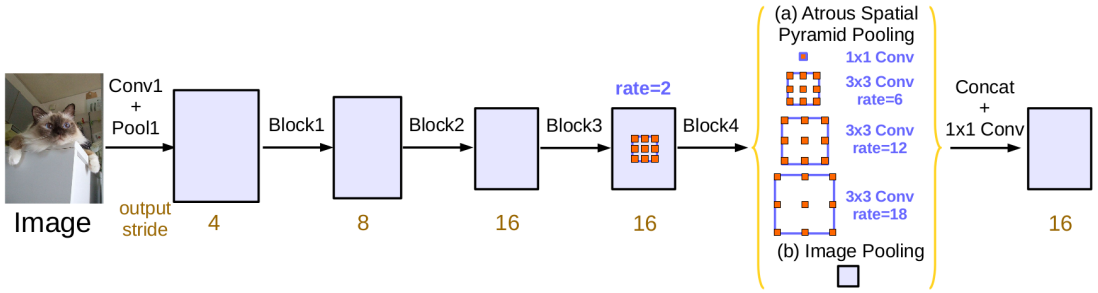
4.3 DeepLabv3

In this version of DeepLab, the contributions include improvements to the context module, and the use of batch normalization. Batch Normalization is applied to every layer in the context module and the parameters of the batch normalization layers are trained.

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



(a) cascade module in DeepLabv3



(b) ASPP module in DeepLabv3

Figure 4.4: Illustration of two different context modules used in deepLabv3 [10].

repetition upto three times improves performance. The cascade module is illustrated in Figure 4.4a. The ASPP module used in DeepLabv3 is similar to the one used in DeepLabv2. However, the difference now is that the ASPP module uses five branches. The first four branches perform 1×1 convolution, and three 3×3 convolutions with atrous rates 6, 12 and 18. The 5th branch provides image level features by performing global average pooling on the last feature maps of the model. The resulting channels are concatenated and projected to a different channel space using 1×1 convolution.

4.4 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. Xception, MobileNetv2 and ResNet-101 are used as encoders out of which this project only considers Xception and MobileNetv2 for their resource efficiency. The major differences in DeepLabv3+ is the use of a decoder, the use of atrous separable convolutions in both the encoder and decoder and the adaption of MobileNetv2 and Xception as network backbones (encoders).

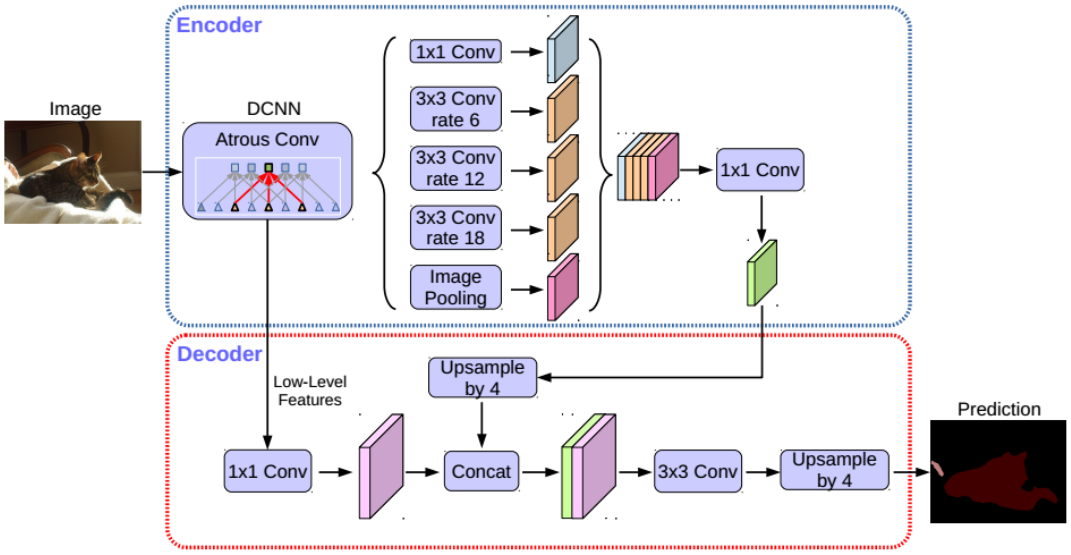


Figure 4.5: An illustration of DeepLabv3+ architecture. The encoder extracts features at different scales and the decoder refines object boundary delineation [11].

The authors call the ratio of input resolution to the output resolution before global average pooling as the output stride. DeepLabv3 is designed to have an output stride of 16. To bring the prediction to the original image resolution, the final features are upsampled by a factor of 16 using bilinear interpolation. This is considered by the authors as a naive decoder module. Instead of this naive approach, the authors propose the use of a better decoder module. The final encoder features are first upsampled by a factor of 4, and are concatenated with the low level features from the encoder with same dimensions. The number of channels in the low level features are first reduced using a 1×1 convolution. A 3×3 convolution convolves over this concatenated features to refine the features and is later followed by an upsampling of 4 to lead to the final prediction. The architecture of DeepLabv3+ is depicted in Figure 4.5.

The MobileNetv2 and Xception encoders make use of depthwise separable convolutions to improve resource efficiency. Section 4.5 and Section 4.6 provide details regarding MobileNetv2 and Xception architectures respectively. In DeepLabv3+, the authors use atrous convolution instead of depthwise convolution in depthwise seperable convolution. The authors call this type of convolution as atrous separa-

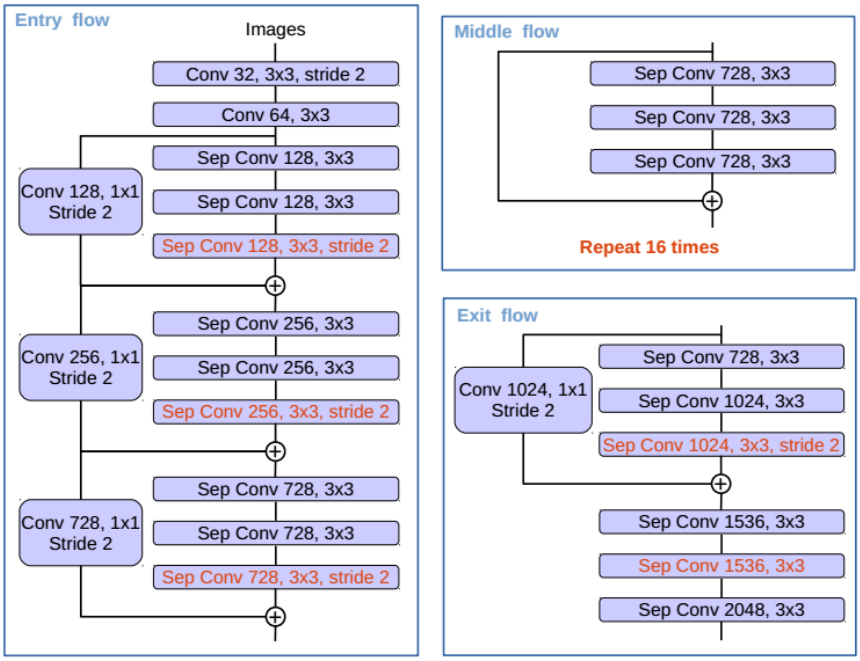


Figure 4.6: Modified Xception architecture used as encoder in DeepLabv3+. The max pooling operations in the original Xception architecture are replaced with depthwise separable convolutions. Batch normalization and ReLU activation is applied after each 3×3 depthwise convolution [11].

ble convolution and state that these convolutions can be used to extract feature maps at arbitrary resolution. The authors use a modified version of the Xception architecture as depicted in Figure 4.6 as one of the network backbones.

4.5 MobileNetv2

The MobileNetv2 architecture is designed to work on mobile and embedded devices where computational resources are limited. The authors state that their main contribution is the use of a novel layer module called the inverted residual with linear bottleneck.

Depthwise separable convolutions, known for its efficiency, is used in this work. Standard convolution layers are replaced with two layers where the first layer performs depthwise convolution and the second layer performs pointwise convolution. A depthwise convolution layer uses a single fiter per input channel to perform convolution. The pointwise convolution layer consists of 1×1 convolutions

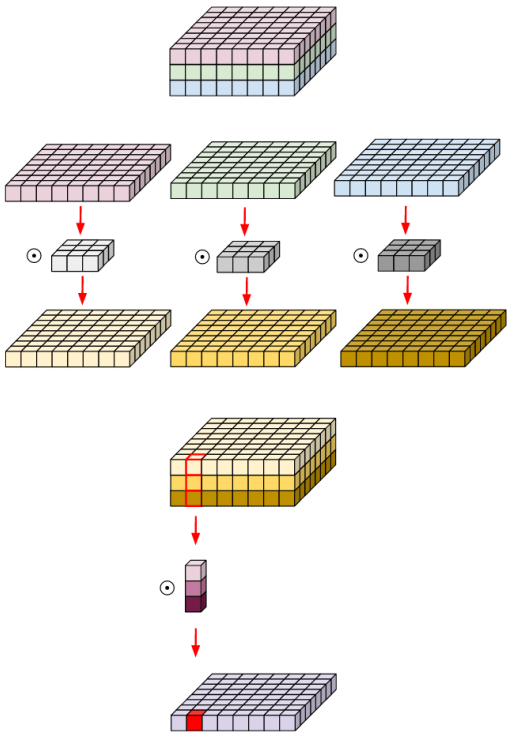
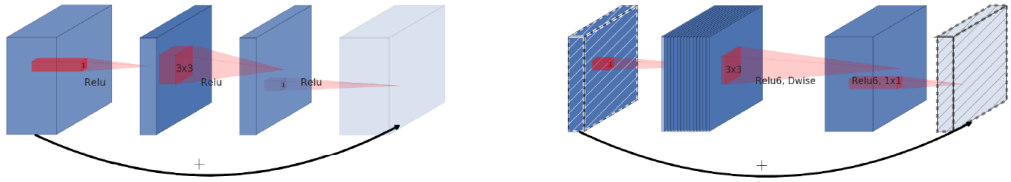


Figure 4.7: An illustration of depthwise separable convolution with 3 input channels and 1 output channel. The first row shows the three input channels. The next three rows illustrate three separate 3×3 convolutions convolving over one of the input channels. The fifth row shows the resulting feature maps of the three convolutions being stacked upon each other. Rows 2 to 5 together is the depthwise convolution. The sixth row shows a pointwise convolution convolving over the output of depthwise convolution to get the final output channel [6].

which perform weighted linear combination on the input channels and projects them to a new channel space. This factorization of standard convolution layer into two separate depthwise and pointwise layers leads to roughly k^2 times reduction in computation cost where k is the kernel size of the convolutional filter. Figure 4.7 illustrates depthwise convolution. In this case, pointwise convolution performs dimensionality reduction as the number of output channels is less than number of input channels. However, if more than 3 pointwise convolutions are used, dimensionality of output channels can be increased.

In the original residual block [19], first a 1×1 convolution is used to reduce the number of channels. On this reduced number of channels 3×3 standard convolution



(a) Residual block (b) Inverted residual block

Figure 4.8: Illustration of the original residual block and the inverted residual block used by MobileNetv2 [33].

is done which is followed by a 1×1 convolution which now expands the feature maps to have the same number of channels as the input to the block. A skip connection is then introduced through which the input channels is added to the output channels of the residual block. The skip connection provides a layer with access to earlier activations and also helps prevent the vanishing gradients problem.

The authors of MobileNetv2 propose the use of inverted residual block which takes advantage of depthwise seperable convolutions. This block consists of a narrow bottleneck layer followed by a 1×1 convolution which performs expansion of number of channels. Depthwise convolution is performed on the expanded input channels followed by a pointwise convolution which brings down the number of channels to create the next bottleneck layer. A skip connection is then added between the input bottleneck and the output bottleneck. The original and inverted residual blocks are depicted in Figure 4.8. The authors use linear activation in the bottleneck layers and hypothesize that it is better than non-linear activation.

This hypothesis is based on the notion that the ReLU activation layer used, leads to information loss due to loss of values less than 0. However, loss of too much information due to the induced nonlinearity by the activation can be prevented by the use of linear activation in the bottleneck layer. The authors show that a linear activation is empirically better than a non linear activation in the bottleneck layer.

The inverted residual block has less number of parameters than the residual block and with linear bottleneck is shown to achieve state-of-the-art performance. The inverted residual block is visualized in the architecture of MobileNetv2 in Figure 4.9.

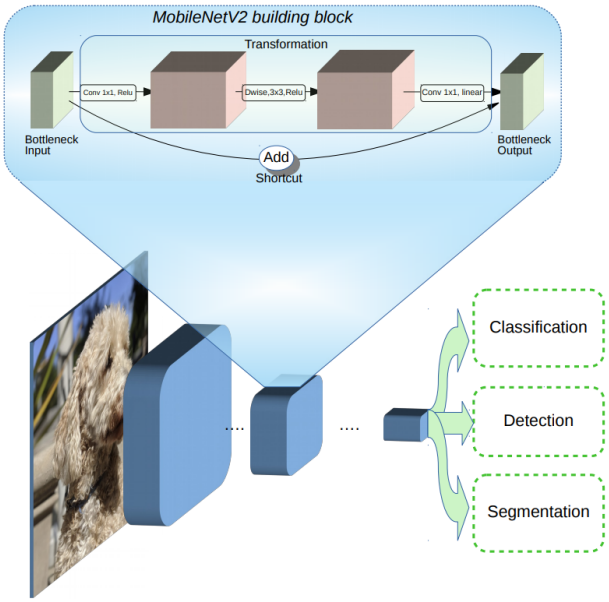
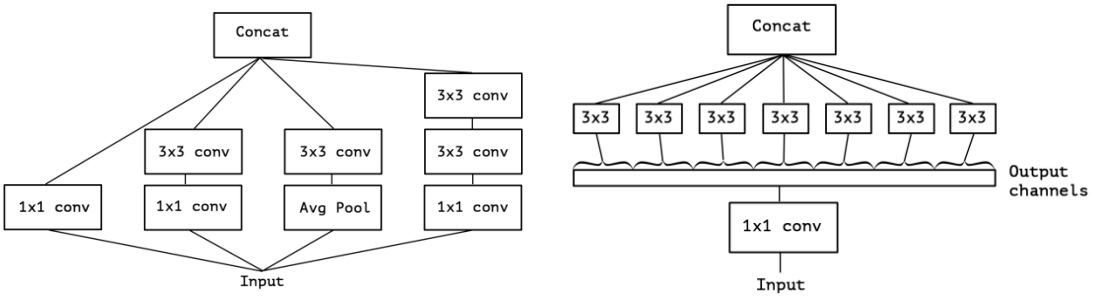


Figure 4.9: The building block of MobileNetv2. The compressed representation obtained could be used for tasks such as image classification, object detection and semantic segmentation [32].

4.6 Xception

The Xception architecture is derived based on two hypothesis made by the authors. One hypothesis is based on the inception architecture and the other is a stronger version of the inception hypothesis. Standard convolution layers has 3D convolution kernals which maps both spatial correlations and cross-channel correlations at the same time. Here spatial correlations is obtained by mapping correlations across the height and the width of every channel separately. Cross-channel correlations is obtained at every spatial location across the height and width of the channels, from each of the channels in the input.

In the Inception architecture, an inception module consists of 4 branches operating on the same input feature maps. The different branches of the inceptionv3 module is shown in Figure 4.10a. In the second branch with 1×1 convolution followed by a 3×3 convolution, the 1×1 convolution calculates the cross-channel correlation between the input channels and performs dimensionality reduction by reducing the number of input channels. Next, the 3×3 convolution is a standard convolution which looks for both spatial and cross-channel correlations in this



(a) Inceptionv3 module

(b) Extreme version of inception module

Figure 4.10: Illustration of the inception v3 module and an extreme version of the inception module. In the extreme version, each of the 3×3 convolution operates on a single output channel of 1×1 convolution [12].

reduced input channel space. Since this inception module has lead to state-of-the-art classification results, the authors hypothesize that the cross-channel correlations and spacial correlations are decoupled and can be mapped separately. This is evidently based on the fact that the inception module partially handles spatial and cross-channel correlations in a decoupled fashion by using a 1×1 convolution followed by a 3×3 convolution.

Based on this hypothesis, the authors propose that the two correlations can be mapped completely independent of each other leading to a stronger hypothesis. An illustration of this stronger hypothesis can be seen in Figure 4.10b. The Xception network is based on this stronger hypothesis and is named "Xception" as the architecture is an extreme version of inception. The complete decoupling of the two correlations, could be obtained using depthwise separable convolution, as evidently, this type of convolution first performs depthwise convoltion which handles the spatial correlation and then performs pointwise convolution which handles cross-channel correlation.

The authors show that the use of depthwise separable convolutions enables the Xception architecture to achieve a marginally better top-1 and top-5 accuracy than the Inception V3 architecture despite the fact that Xception has less number of parameters. This experimental observation suggests that Xception is resource efficient and is also powerful in terms of "learning capacity".

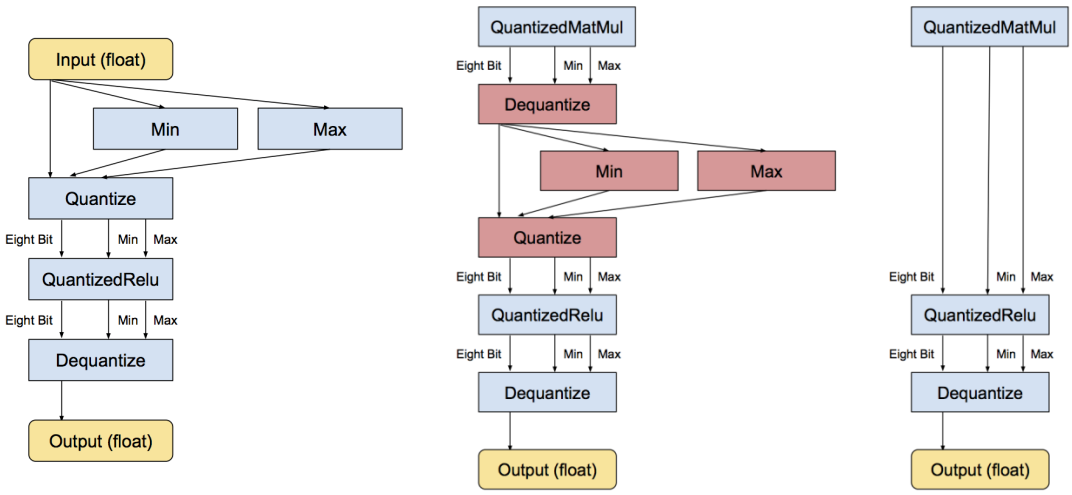
4.7 Quantization

This work makes use of the quantization method available in tensorflow source repository [1]. "eightbit" mode is used to obtain a quantized tensorflow inference graph. The blog in [39] provides a description of this quantization procedure.

Neural networks, in general, learn features which are robust to noise in the input image such as changes in illumination. As an extension, it is possible to speculate that loss in precision induced by low-precision calculations is ignored by a neural network as another source of noise. High precision calculation, though necessary to handle gradient propagation during training, might not be necessary during inference. Quantizing a neural network calculations to low-precision could therefore be a way to reduce computation cost without much loss in accuracy in order to facilitate usage on an embedded device. More specifically, quantizing calculations to eight bit calculations would make a neural network faster and reduce storage requirements on embedded devices.

The quantization algorithm replaces common operations such as convolution, activation functions and pooling operations with equivalent quantized operations. The input data is converted from float to eight bit precision by adding a subgraph as illustrated in Figure 4.11b. The algorithm also identifies sequences of quantized operations and removes subgraphs which perform redundant conversion as illustrated in Figure 4.11a. The overall maximum and minimum possible values of weights and activation tensors are representated by the lowest and highest quantized values respectively which is 0 and 255 for eight bit. The rest of the values of the tensors are then represented within the range of eight bits. This format of quantization brings certain advantages such as the ability to represent arbitrary ranges and negative numbers, and the linear spread of values facilitates multiplication. The minimum and maximum values are determined by the "QuantizeDownAndShrinkRange" operator which analyzes and rescales the tensor to eight bit in a manner that the range of eight bit values is used effectively.

[1] `https://github.com/tensorflow/tensorflow/blob/master/tensorflow/tools/quantization/quantize_graph.py`



(a) (b)

Figure 4.11: This figure shows illustrations of the quantization procedure available in tensorflow source. (a) Replacing an operation with a subgraph which performs qantization, corresponding quantized operation and dequantization. (b) The figure on the left shows a redundant subgraph (indicated in red) in between two quantized operations and the figure on the right shows the graph obtained after removing the redundant subgraph [39].

5

Dataset creation

5.1 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 Figure 5.1.

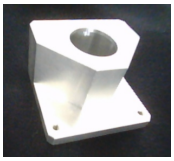
Each of the objects were taken individually, placed on 3 different backgrounds and 30 images were taken. This lead to a total of 540 images which were to be manually labeled. Since, every pixel of the images needs to be labeled, the process of manual annotation would be time consuming. Therefore, a decision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

Details regarding the labeling tools and the description of the labeling process can be found in appendix A.

(a) axis [7]

(b) bearing [7]

(c) bearing box AX01 [7]

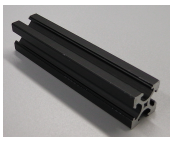


(d) bearing box AX16

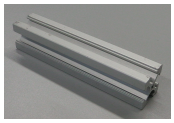
(e) container blue [7]

(f) container red [7]

(g) distance tube [7]



(h) F20_20_B [7]



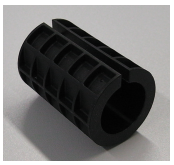
(i) F20_20_G [7]

(j) M20 [7]

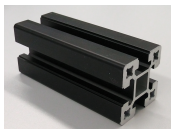
(k) M20_100 [7]

(l) M30 [7]

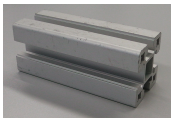
(m) motor [7]



(n) R20 [7]



(o) S40_40_B [7]



(p) S40_40_G [7]

(q) em_01

(r) em_02

Figure 5.1: Different objects required in the dataset

5.2 Artificial image generation algorithm

5.2.1 Motivation

- Manually labeling 540 images with the described labeling process in appendix A 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.

5.2.2 Process of artificial image generation

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.

5.2.3 Generator options

A number of arguments can be configured to control the generation process. Configuration of generator options is possible through command line GUI. Description of the generator options is provided in Table 5.1 and further details can be found in appendix A.

5.2.4 Sample results

Sample results of the artificial image generation algorithm can be seen in Figure 5.2. These images are generated using different backgrounds and hence the dataset is referred to as variety of backgrounds dataset. The bounding box in the label visualization image 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 [37].

5.2.5 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 consuming. Hence, the "google-images-download" [38] script was used to auto download images. The search keywords used to obtain the background images are listed in appendix A. From the downloaded images, the required number of images for each dataset split were selected. For the white backgrounds dataset, many different search keywords were tried. 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.

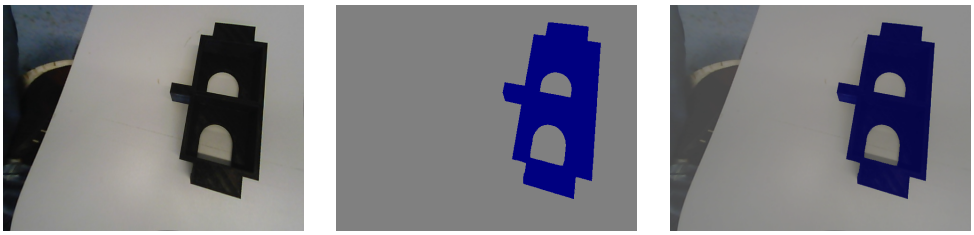
5.2.6 Notable features of the artificial image generator

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

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 5.1: Description of generator options

Figure 5.2: Sample results produced by the artificial image generation algorithm for the variety of backgronds dataset. 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.



(a) Image containing object em_01

(b) Corresponding segmentation mask

(c) Corresponding overlay visualization

Figure 5.3: This figure shows examples of two different types of visualization, "mask" and "overlay". The third type "preview" can be seen in figure 5.2

- 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 Figure 5.3 and in Figure 5.2.

5.2.7 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 default scale range of [0.24, 1.2] is retained for all objects except for the "distance_tube". The scale range of "distance_tube" was set to [1.1, 2.0] for generating training artificial images and to [0.6, 1.2] for generating validation and test artificial images. This is because the lower limit of 0.24 in the default scale range zooms down "distance_tube" in a manner that it is no longer perceivable which is not desired.

5.3 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.

5.3.1 Motivation

Looking into the objects present in the dataset, it is apparent 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 Figure 5.4 and in Figure 5.5. 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 5.4: A 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 5.5: A viewpoint of bearing box ax01 and bearing box ax16 where they appear similar in shape.

5.3.2 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 Table 5.2.

- **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 Table 5.2. 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 Table 5.2.

- **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 Table 5.2.

5.3.3 White backgrounds 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

Label Value	atWork_ full objects	atWork_size_ invariant objects	atWork_similar_ shapes objects	atWork_ binary objects
0	background	background	background	background
1	f20_20_B	f_s20_40_20_40_B	f_s20_40_20_40_B_G	foreground
2	s40_40_B	f_s20_40_20_40_G	m20_100	-
3	f20_20_G	m20_100	m20_30	-
4	s40_40_G	m20_30	r20	-
5	m20_100	r20	bearing_box	-
6	m20	bearing_box	bearing	-
7	m30	bearing	axis	-
8	r20	axis	distance_tube	-
9	bearing_box_ax01	distance_tube	motor	-
10	bearing	motor	container	-
11	axis	container_box_blue	em_01	-
12	distance_tube	container_box_red	em_02	-
13	motor	em_01	-	-
14	container_box_blue	em_02	-	-
15	container_box_red	-	-	-
16	bearing_box_ax16	-	-	-
17	em_01	-	-	-
18	em_02	-	-	-

Table 5.2: This table lists the objects in each variant and its corresponding label value.

"white backgrounds dataset". Sample visualizations of artificial images generated for this dataset can be seen in Figure 5.6.

5.4 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 class appears in the training set of the variety of backgrounds dataset. This provides insights regarding whether or not all the objects are represented equally by the dataset. If an object occupies pixels which is significantly greater than the pixels occupied by all other objects, this object can be considered to dominate the dataset. Intuitively, a segmentation model would favor a dominant object as classifying this object

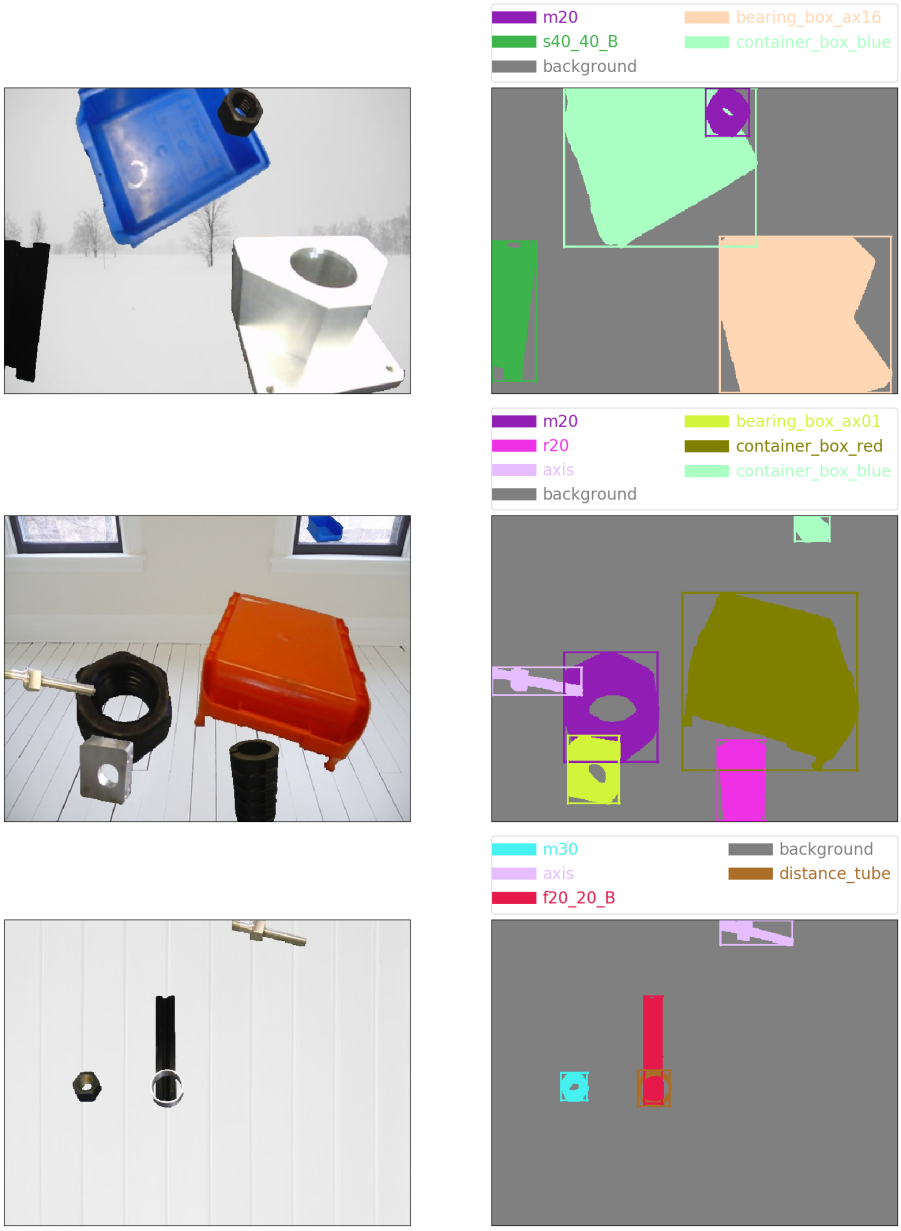


Figure 5.6: Sample results produced by the artificial image generation algorithm for the "white backgrounds dataset".

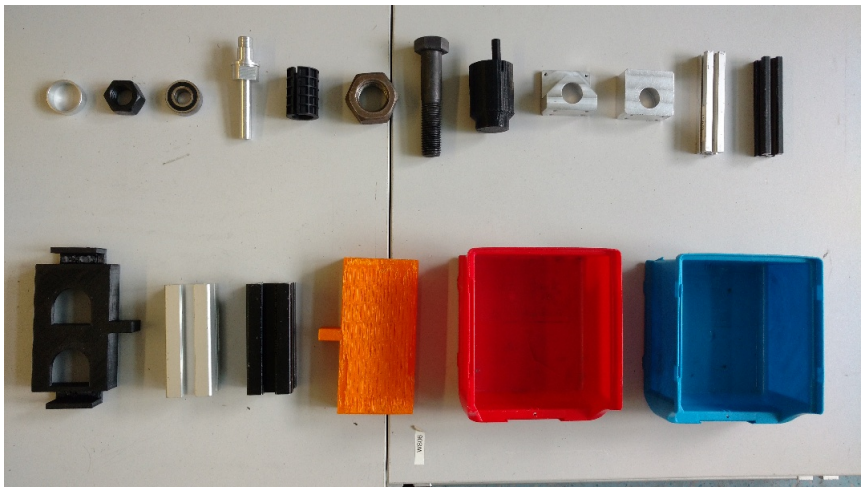


Figure 5.7: This figure shows all the 18 objects in the dataset arranged in increasing order of surface area from top left to bottom right. First row from left: "distance_tube", "m20", "bearing", "axis", "r20", "m30", "m20_100", "motor", "bearing_box_ax16", 'bearing_box_ax01", "f20_20_B", "f20_20_G". Second row from left: "em_01", "s40_40_B", "s40_40_G", "em_02", "container_box_red", "container_box_red".

correctly would take the model closest to a global minimum. On this regard, sorting objects in terms of their dominance in the dataset would help provide clues to why a segmentation model segments certain objects better than others.

5.4.1 Surface area of the objects

In order to comprehend the reasons as to why an object constitutes a certain percentage of pixels in the training set, 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. The surface area of all the objects in the dataset is obtained using 3D CAD models available in the atWork github repositories [1] and [2]. The surface areas are listed in Table 5.3.

Object name	**Surface area** (cm^2)	**Percentage of pixels (%)**	**Class count**
f20_20_B	158.36	0.2948	1126
s40_40_B	426.77	0.4218	1118
f20_20_G	158.36	0.2683	1177
s40_40_G	426.77	0.896	1245
m20_100	90.00	0.4577	1173
m20	37.59	0.2981	1120
m30	83.84	0.2921	1087
r20	80.82	0.3648	1117
bearing_box_ax01	114.83	0.8653	1369
bearing	59.51	0.2526	1195
axis	67.30	0.4214	1190
distance_tube	21.39	0.2321	1028
motor	112.31	0.486	1079
container_box_blue	1244.50	2.6529	1680
container_box_red	1244.50	2.2299	1375
bearing_box_ax16	106.38	0.5657	1313
em_01	372.85	1.224	1271
em_02	544.54	1.5624	1261

Table 5.3: This table lists the surface area occupied by all the 18 objects and their corresponding percentage of pixels and class count on the atWork_full variant of the variety of backgrounds dataset.

5.4.2 Percentage of pixels and class count

The total pixels occupied by each class in the training set 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 lower plot in Figure 5.8a shows normalized values of surface area and percentage of pixels of objects in the atWork_full variant plotted as histograms. The bars are arranged in the increasing order of surface area. The percentage of pixels also seems to increase along with surface area but is not consistent. However, when every 3 classes are combined starting from "distance_tube", and plotted separately shown in the upper plot of 5.8a, an apparent pattern appears which shows an increasing trend followed by percentage of pixels in propotion to surface area. A similar inference could be made for atWork_size_invariant and atWork_similar_shapes variant from plots in Figure 5.8b and Figure 5.8c respectively.

An indirect reason for this apparent increase of percentage of pixels in propotion to surface area could be attributed to the generator options used in the artificial image generation algorithm. The artificial image generator ignores scaled objects which are too small and attempts to generate objects which are bigger. This is done to avoid objects which tend to be not visible in the produced artificial images. As a result, objects which are bigger are more likely to survive the artificial image generation process and appear more in the dataset.

An important inference from this data analysis is that a segmentation model is likely to ignore smaller objects such as "distance_tube" and is likely to learn bigger objects such as "container_box_red" better.

5.5 Meta-data of the dataset

Meta-data of the dataset is provided in Table 5.4. 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.

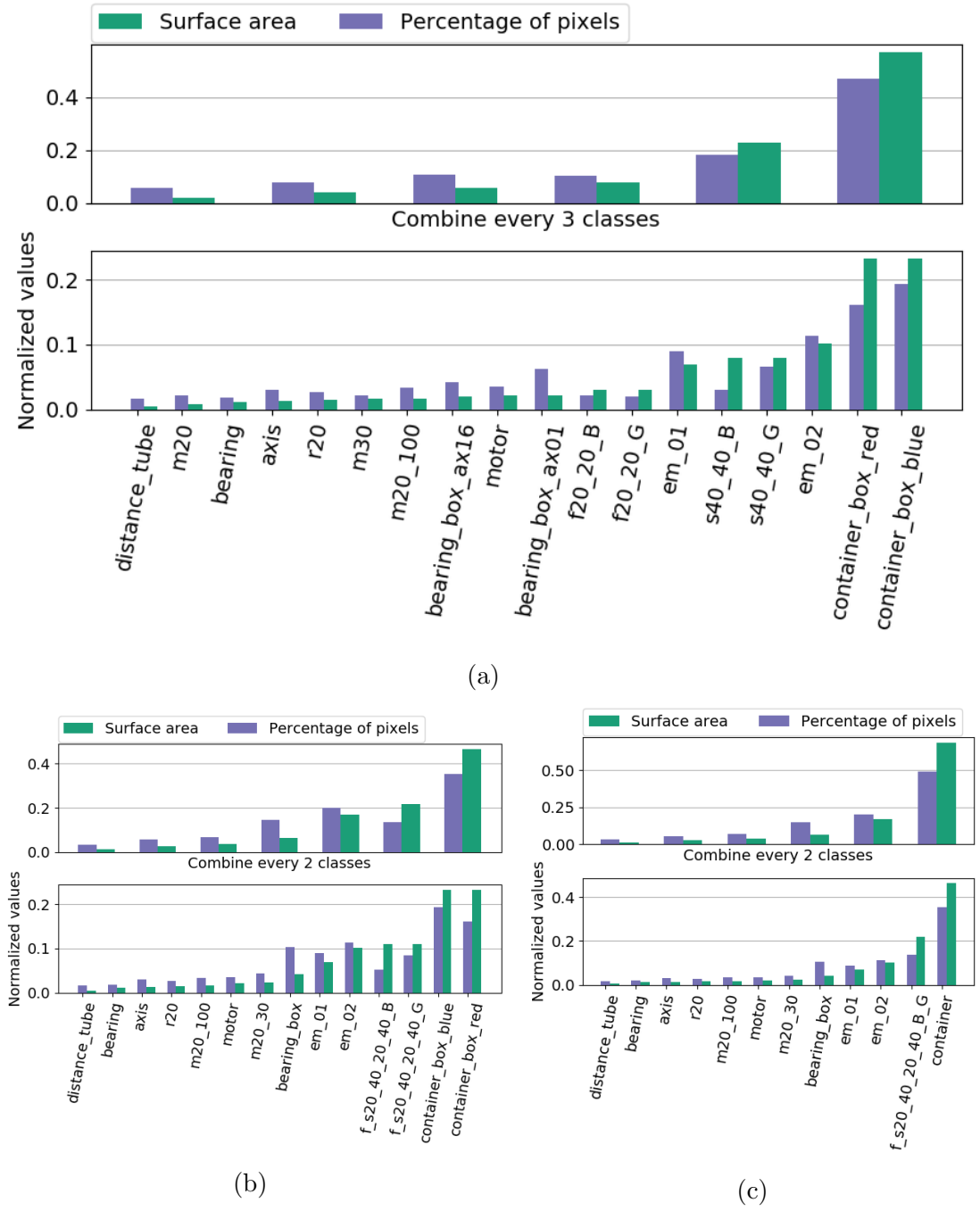


Figure 5.8: This plot shows that the percentage of pixels occupied by objects in the variety of backgrounds dataset is related to the real world surface area of the objects. The plots are for different dataset variants: (a): atWork_full variant, (b): atWork_size_invariant, and (c): atWork_similar_shapes.

	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
Artificial Images	7104	870	870
Total Images	7500	942	939

Table 5.4: Meta-data of all 4 variants of the variety of backgrounds and the white backgrounds dataset.

5.6 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 [28] and the Synthia dataset [31], takes a different approach. Unlike [28], 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 [31]. 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 artificial image generation algorithm is written in python and is independent of the MATLAB ImageLabeler app. This can be improved by including a way to start artificial image generation right from the ImageLabeler.

6

Experimental Evaluation

Since the major contribution of this work is the creation of the dataset, the experiments are focused on validating the effectiveness of the dataset. This chapter is divided into 9 sections. Section 6.1 called "About the metrics" describes the different metrics used for evaluation. Section 6.2 called "Comparing dataset variants" compares the performance of DeepLabv3+ across the 4 variants of the dataset. Section 6.3 called "Comparing DeepLabv3+ backbones" compares the two network backbones MobileNetv2 and Xception of DeepLabv3+. Section 6.4 called "Training with different data" compares the performance of DeepLabv3+ on the real validation set when different training data combinations such as only training with real data and training with a combination of real and artificial data is used. Section 6.5 called "Comparing individual classes" compares the performance of DeepLabv3+ with MobileNetv2 network backbone on different classes of objects using confusion matrix and individual class IOUs. Section 6.6 called "Comparing learning rate policies" compares two different learning rate decay policies considered for this work. Section 6.7 called "Effects of class balancing" describes the attempt made to use a weighted loss function to prevent DeepLabv3+ model from favoring dominant classes. Section 6.8 called "Effects of quantizing the inference graph" looks into the differences between full precision models and corresponding low precision models. Section 6.9 called "Discussions" provides conclusions and further insights gained through the experiments.

6.1 About the metrics

The metrics used for evaluation are Mean Intersection Over Union (mIOU), inference time, number of parameters, floating point operations (FLOPS) and occupied disk memory.

- **mIOU**: Mean Intersection Over Union is an accuracy metric used for semantic segmentation. IOU of a particular class, is the ratio between pixels shared between the ground truth and the prediction to the total pixels of the corresponding class belonging to the ground truth and prediction.

$$IOU = \frac{ground\_truth \cap prediction}{ground\_truth \cup prediction}. \tag{6.1}$$

  mIOU is the mean of IOUs of all classes in the dataset. IOU could also be interpreted as shown in equation (6.2) on the pixel level where, TP denotes True Positives which is the total number of pixels correctly predicted, FP denotes False Positives which is the total number of pixels which does not belong to the particular class according to the ground truth but is predicted as belonging to the class and FN denotes False Negatives which is the total number of pixels not belonging to the class but is predicted as belonging to the class.

$$IOU = \frac{TP}{TP + FP + FN} \tag{6.2}$$

  When the ground truth and predictions have no pixels intersecting for a class, the corresponding class IOU will be 0 percent. Similarly, if there is perfect union between the ground truth and the prediction, the corresponding class IOU will be 100 percent.

- **Inference time**: The time taken, in seconds, by a segmentation model to calculate the output predictions for one input image.

- **Number of parameters**: The total number of trainable parameters in a segmentation model which are updated during training. Convolution kernel weights, biases, batch normalization parameters are examples of trainable parameters.

- **Floating point operations (FLOPS)**: The total number of floating point operations performed when one image goes through the network in one forward pass.
- **Occupied disk memory**: The total storage space occupied by the network in Mega Bytes (MB).

6.2 Comparing dataset variants

- **Objective**: The objective of this experiment is to compare the performance of DeepLabv3+ on the different dataset variants.
- **Expected result**: DeepLabv3+ is expected to obtain a higher mIOU 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 MobileNetv2 network backbone and the Xception network backbone are evaluated on all variants of variety of backgrounds and white backgrounds dataset. From Figure 6.1, it is evident that the mIOU 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 at arbitrary scales and are subject to differences in illumination, the real world differences between such similar objects become insignificant in the dataset. Thus, the mIOU obtained on 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, DeepLabv3+ achieves better mIOU on these variants. The atWork_binary variant requires the DeepLabv3+ to only distinguish foreground from background leading to the highest mIOU. Evidently, the stated inferences are independent of the network backbone used by DeepLabv3+. The mIOU values obtained are tabulated in Table 6.1.

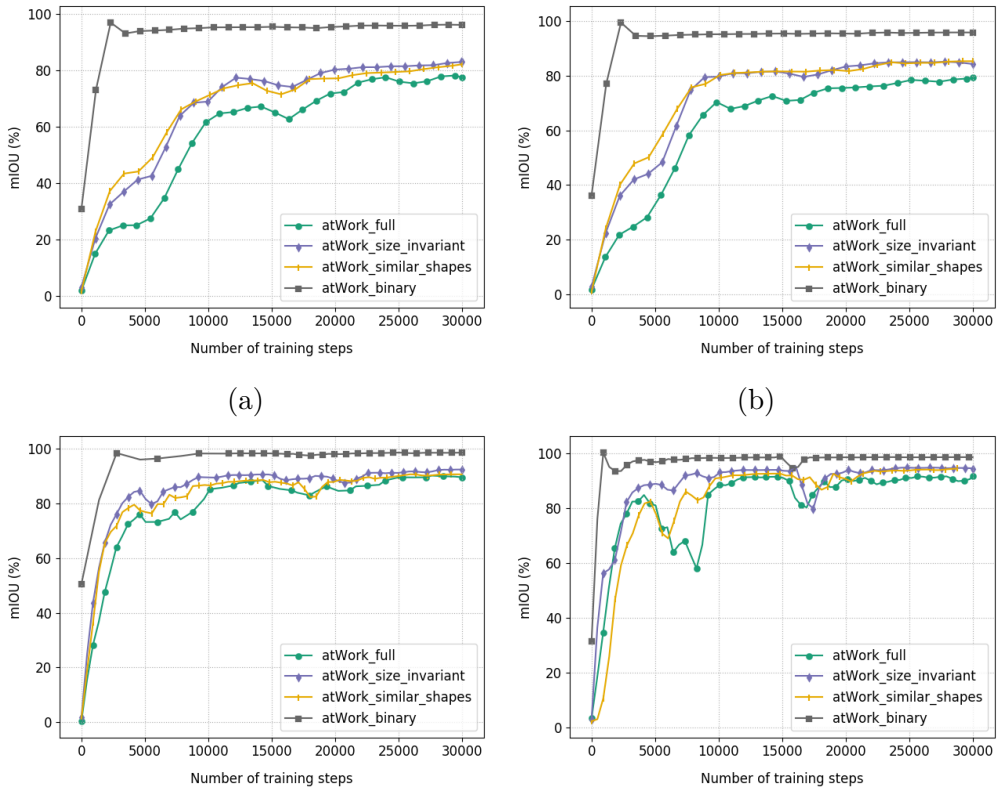


Figure 6.1: mIOU of Deeplabv3+ on the different dataset variants. (a) MobileNetv2 network backbone on variety of backgrounds dataset, (b) MobileNetv2 network backbone on white backgrounds dataset, (c) Xception network backbone on variety of backgrounds dataset, (d) Xception network backbone on white backgrounds dataset.

Dataset variant	**mIOU in %**			
	MobileNetv2 backbone		Xception backbone	
	VB	WB	VB	WB
atWork_full	77.47	79.26	89.63	91.59
atWork_size_invariant	83.10	84.29	92.47	94.27
atWork_similar_shapes	82.10	85.33	90.71	94.33
atWork_binary	96.06	95.83	98.68	98.47

Table 6.1: This table lists the mIOU obtained by DeepLabv3+ with MobileNetv2 and Xception network backbones on 4 dataset variants of VB: variety of backgrounds dataset and WB: white backgrounds dataset.

6.3 Comparing DeepLabv3+ backbones

- **Objective**: The objective of this experiment is to compare the mIOUs obtained by DeepLabv3+ with MobileNetv2 and Xception network backbones on each of the datasets and its variants.

- **Expected result**: The Xception network backbone is expected to obtain higher mIOU because of the higher number of learnable parameters in comparison with the MobileNetv2 network backbone. In essence, the Xception network backbone has more "learning capacity" than the MobileNetv2 network backbone leading to the ability to learn a better decision boundary.

- **Inference from the results**: The results obtained are shown in Figure 6.2. Across all the dataset variants, the Xception network backbone achieves higher mIOU than the MobileNetv2 network backbone consistently. Another inference is that the models trained and validated on the white backgrounds dataset achieves slightly higher mIOU than corresponding models trained on the variety of backgrounds dataset. This could be because the model has to handle less variations in terms of backgrounds in the white backgrounds dataset as the backgrounds in this dataset are similar to each other.

6.4 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.

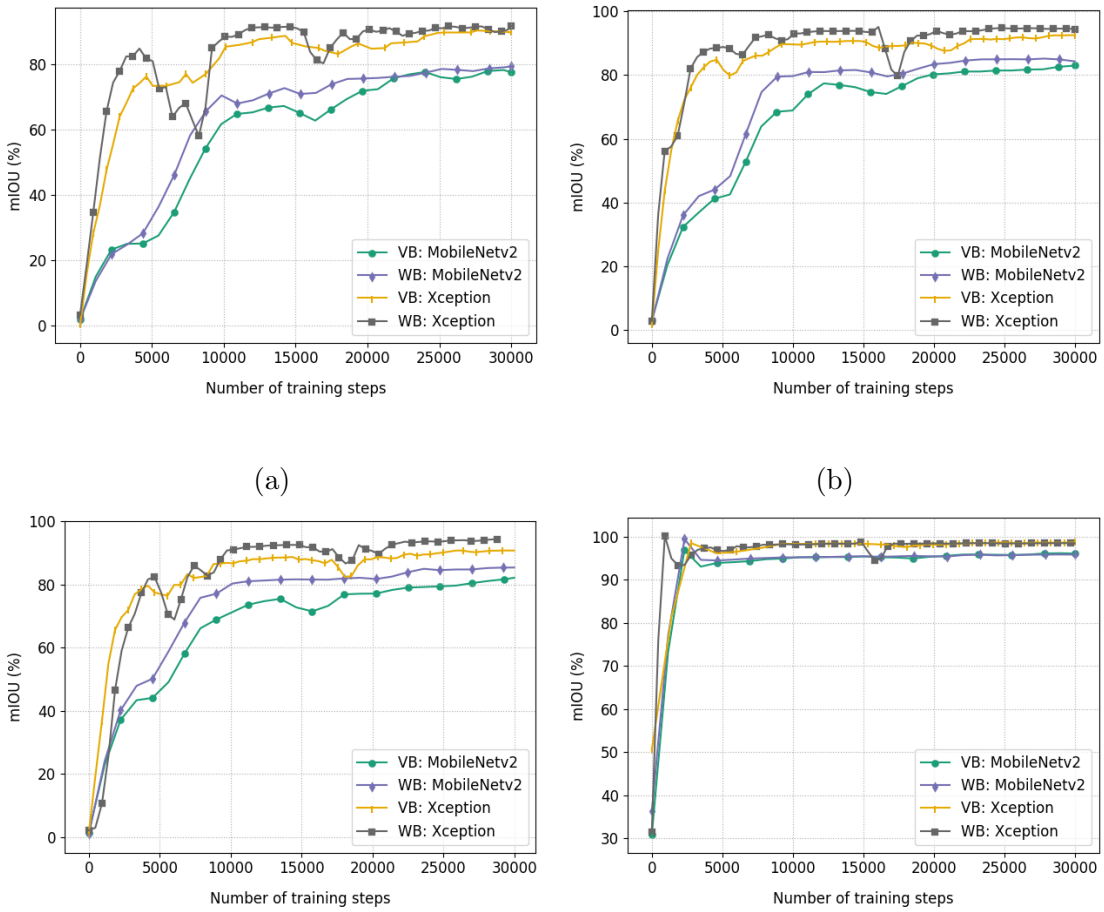


Figure 6.2: Comparison of mIOU obtained by DeepLabv3+ using MobileNetv2 network backbone vs Xception network backbone on all 4 variants. VB denotes the variety of backgrounds dataset and WB denotes the white backgrounds dataset. The dataset variant is (a): atWork_full, (b): atWork_size_invariant, (c): atWork_similar_shapes and (d): atWork_binary.

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 mIOU. This is based on the notion that the artificial images forces the segmentation model to learn features independent of the background. Also, the model 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 as it does not introduce the segmentation model to real world conditions.

- **Inference from the results**: From the results shown in Figure 6.3 and in Table 6.2, it is evident that in all variants, training with just the real images achieves the best mIOU on the real validation set. This is in contrast to the notion that augmenting with artificial images improves mIOU. This apparent 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 existence 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 because of the need to reduce the labeling cost. Creating real world variations in terms 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 backgrounds.

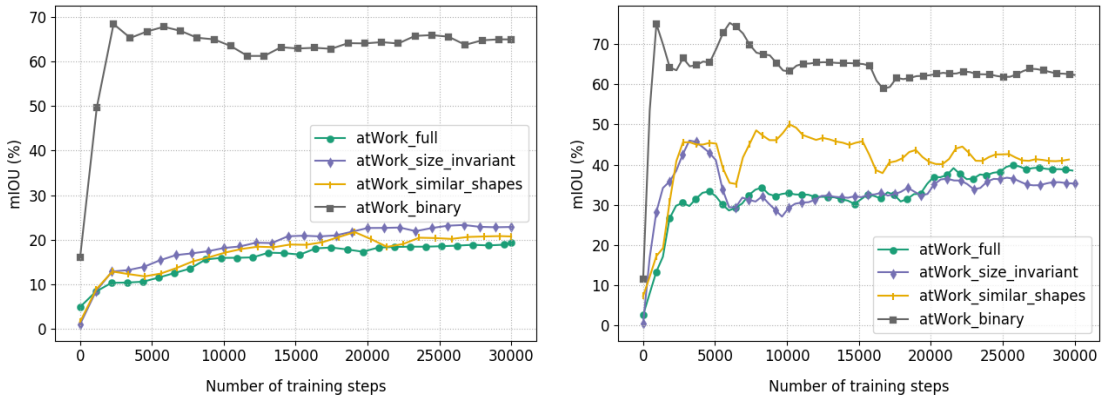
Figure 6.3: This plot shows the mIOUs obtained when training DeepLabv3+ on different training sets. "full" denotes the atWork_full variant, "size" denotes the atWork_size_invariant variant, "shape" denotes the atWork_similar_shapes variant, "binary" denotes the atWork_binary variant.

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.

- **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 artificial data. The results are shown in Figure 6.4 and Table 6.3 summarizes the mIOU values. In this case, training with just the real data consistently obtains low mIOU across all variants using both Xception and MobileNetv2 network backbones. This suggests that the speculations made could be further explored.

Dataset variant	**Backbone**	**mIOU in %**			
		Real	**VB: All**	**WB: All**	**VB: Artificial**
atWork_full	MobileNetv2	83.21	71.72	70.80	40.00
	Xception	87.03	80.26	78.42	45.67
atWork_size_invariant	MobileNetv2	85.01	80.08	77.12	47.76
	Xception	90.84	89.58	87.67	41.58
atWork_similar_shapes	MobileNetv2	79.83	77.33	76.47	43.31
	Xception	92.85	87.76	83.58	43.32
atWork_binary	MobileNetv2	94.33	93.01	90.17	43.29
	Xception	98.19	95.21	94.31	47.91

Table 6.2: This table summarizes the mIOUs obtained when training with different training data and validating on the real validation data. "Real" denotes real training data, "VB: All" denotes real and artificial data of the variety of backgrounds dataset, "WB: All" denotes real and artificial data of the white backgrounds dataset, "VB: Artificial" denotes artificial data of the variety of backgrounds dataset.



(a) (b)

Figure 6.4: This figure shows the mIOUs obtained when trained on real images and validated on artificial images across all 4 variants of the variety of backgrounds dataset. (a) mIOU obtained by MobileNetv2 network backbone, (b) mIOU obtained by Xception network backbone.

Dataset variant	Backbone	mIOU in %
atWork_full	MobileNetv2	19.24
	Xception	38.54
atWork_size_invariant	MobileNetv2	22.83
	Xception	35.27
atWork_similar_shapes	MobileNetv2	20.70
	Xception	41.28
atWork_binary	MobileNetv2	65.03
	Xception	62.32

Table 6.3: This table summarizes the mIOUs obtained by both the network backbones of DeepLabv3+ model when trained on real images and validated on artificial images across all variants of the variety of backgrounds dataset.

6.5 Comparing individual classes

6.5.1 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**: The obtained confusion matrices are shown in Figure 6.5. On all the confusion matrices, the leading diagonal elements have highest values in each row. This is expected and suggests that the model correctly 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

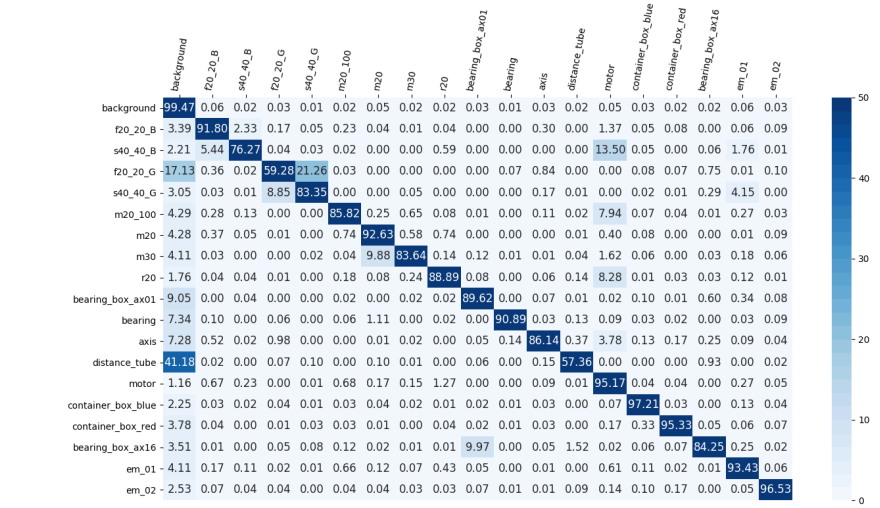
cannot be picked up by the model as no consistent information regarding the object size is available in the dataset. 41.18 percent of pixels in distance tube are confused with background. This could be because the number of pixels occupied by distance tube in the dataset is small in comparison to the 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 attributed 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.

6.5.2 Class IOUs

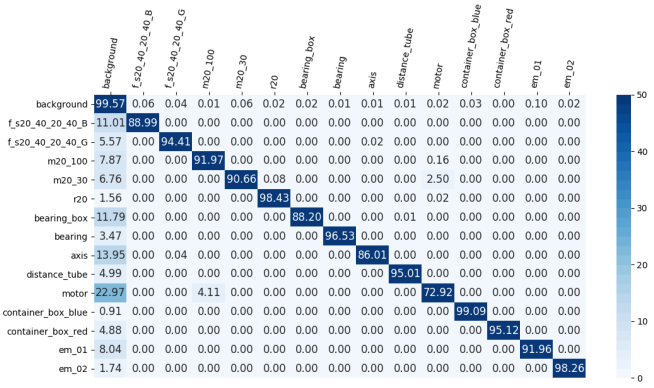
- **Objective**: The objective of this experiment is to look for a relationship between the individual class IOUs and the percentage of pixels occupied by each class in the dataset. In extension, a relationship between class IOUs and real world surface area could also be obtained.

- **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.

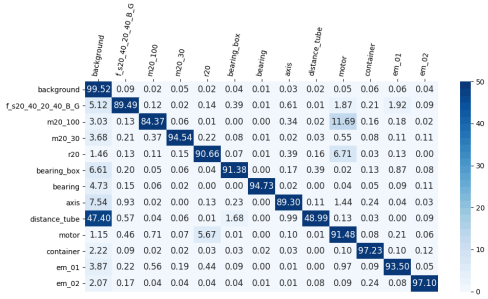
- **Inference from the results**: For each class, the mean over 30000 training steps of class IOU is calculated. Both the percentage of pixels and the class IOU is normalized with respect to the maximum value out of all objects. The classes are arranged in increasing order of percentage of pixels. The plots are shown in Figure 6.6. From the lower histogram plot for atWork_full variant shown in Figure 6.6a, the class IOUs denoted by the green bars do not seem to show an increasing trend. However, when every 3 classes starting from class distance_tube are combined and plotted separately, (shown in the upper graph of Figure 6.6a), the class IOUs show an increasing trend in propotion to percentage of pixels. Similar observations can be made for the other two variants as shown in Figure 6.6b and in Figure 6.6c. Another interesting result is that the segmentation model seems to learn the object "bearing"



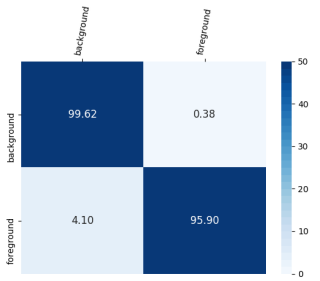
(a)



(b)



(c)



(d)

Figure 6.5: Confusion matrix of DeepLabv3+ with MobileNetv2 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.

well despite the fact that the object only occupies few pixels in the dataset. This could be because of the distinct black ring in between two silver rings present in the bearing. This pattern seems unique and the model probably picks up this pattern with ease.

6.6 Comparing learning rate policies

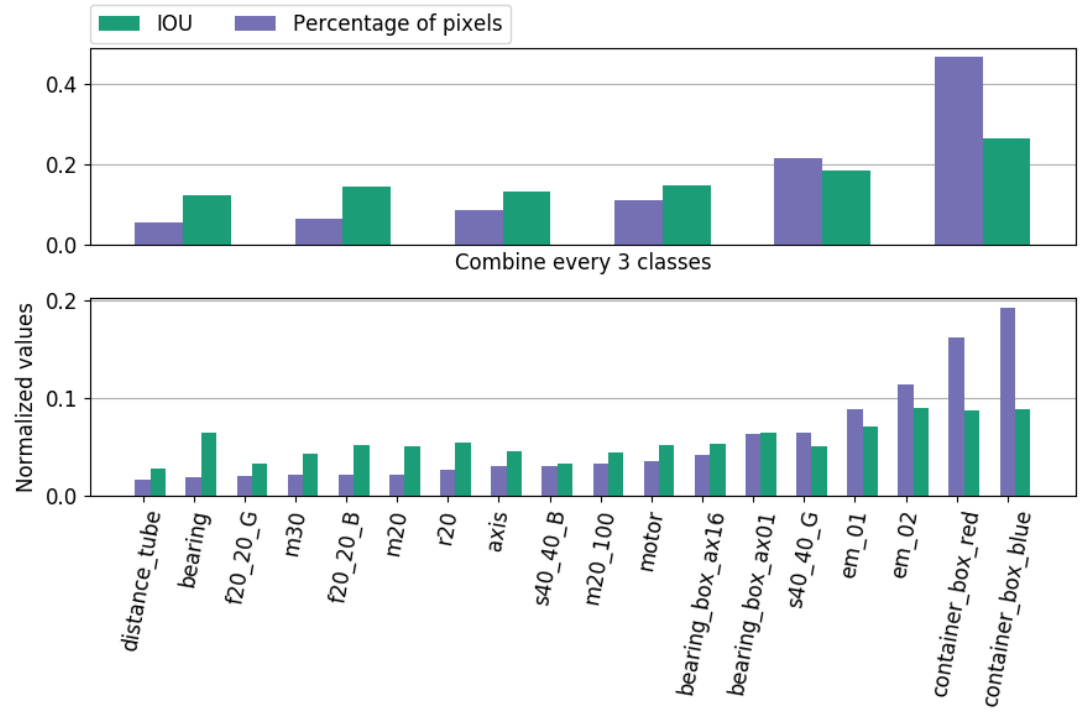
- **Objective**: The objective of this experiment is to compare the cosine restarts [23] learning rate policy with the poly learning rate policy used by DeepLabv3+.
- **Expected result**: Either of the two learning rate policies is expected to result in better Mean IOU.
- **Inference from the results**: DeepLabv3+ with MobileNetv2 backbone is used for this experiment. The mIOU obtained using the two learning rate decay policies are shown in Figure 6.7. Evidently, the cosine restart learning rate policy leads to slightly better mIOU on both the atWork_binary and the atWork_size_invariant variants.

6.7 Effects of class balancing

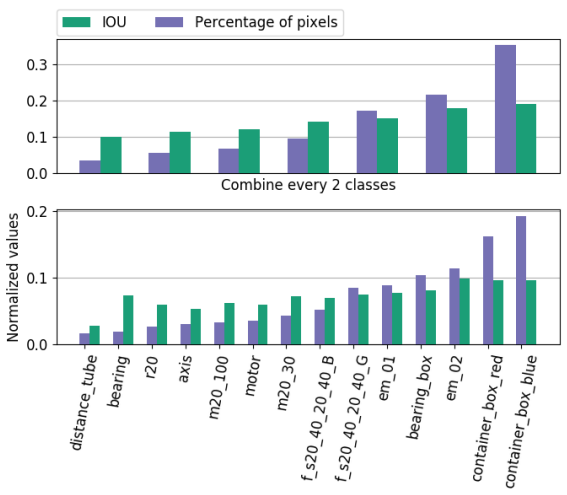
- **Objective**: The objective of this experiment is to prevent the DeepLabv3+ model from giving preference to dominant classes in the dataset. A weight coefficient is determined for each class based on the percentage of pixels occupied by the class in the dataset. The weight coefficients of each class is calculated using median-frequency re-weighting [16] shown in equation (6.3).

$$\alpha_c = median\_freq/freq(c) \tag{6.3}$$

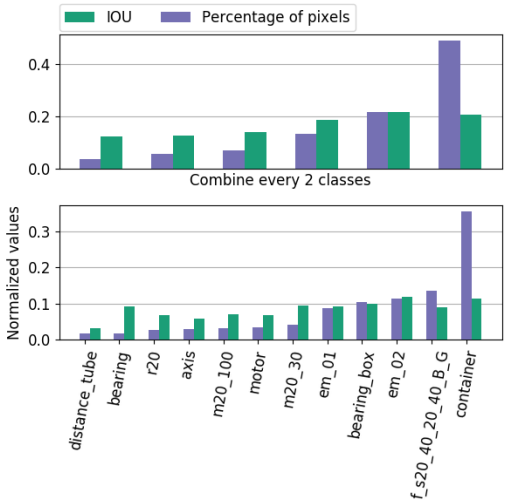
In equation (6.3), α_c is the class coefficients, "median_freq" refers to the median of all class percentage values and "freq(c)" refers to the corresponding class percentage. These weight coefficients are multiplied with the loss term of the corresponding class in the loss function.



(a)

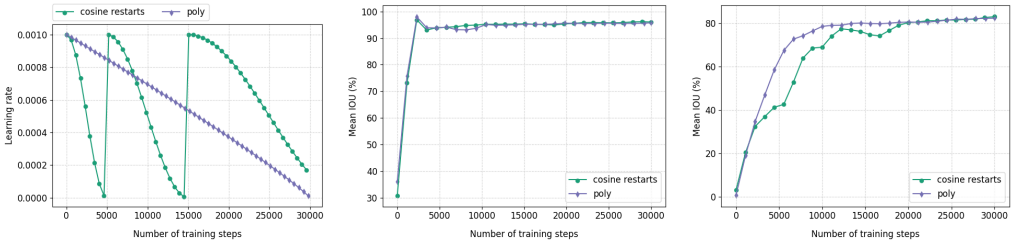


(b)



(c)

Figure 6.6: Individual class IOUs achieved by DeepLabv3+ with MobileNetv2 backbone is plotted with the percentage of pixels occupied on all 4 variants of the variety of backgrounds dataset. Both the individual class IOUs and percentage of pixels are normalized to lie in the range of [0,1]. (a): atWork_full variant, (b): atWork_size_invariant, and (c): atWork_similar_shapes.



(a) (b) (c)

Figure 6.7: Learning rate decay with two different policies 1. cosine restarts and 2. poly is compared. (a): learning rate over 30000 steps with the two decay policies. (b): mIOU on the validation set of atWork_binary variant is 96.06 % with cosine restarts and 95.75 % with poly. (c): mIOU on the validation set of atWork_size_invariant variant is 83.1 % with cosine restarts and 82.24 % with poly.

- **Expected result**: The model is expected to achieve similar class IOUs on all the objects. The overall mIOU is also expected to be at least slightly better than the mIOU obtained without class balancing.
- **Inference from the results**: From Figure 6.8, clearly, the IOU obtained on each class reduces after performing class balancing. This suggests that class balancing using median-frequency re-weighting is undesirable. The root cause for this result is unclear and needs further analysis.

6.8 Effects of quantizing the inference graph

- **Objective**: The objective of this experiment is to compare a model with floating point weights and the corresponding model with fixed point weights in terms of mIOU, occupied disk memory and inference time.
- **Expected result**: The 8bit models are expected to occupy less disk memory and have less inference time in comparison with the corresponding full precision models. In terms of mIOU, the 8bit models are expected to achieve comparable mIOU to the corresponding full precision models despite the reduced "learning capacity" due to the use of a fixed point representation.
- **Inference from the results**: From Figure 6.9b it is evident that across all the variants and network backbones of DeepLabv3+, the 8bit low preci-

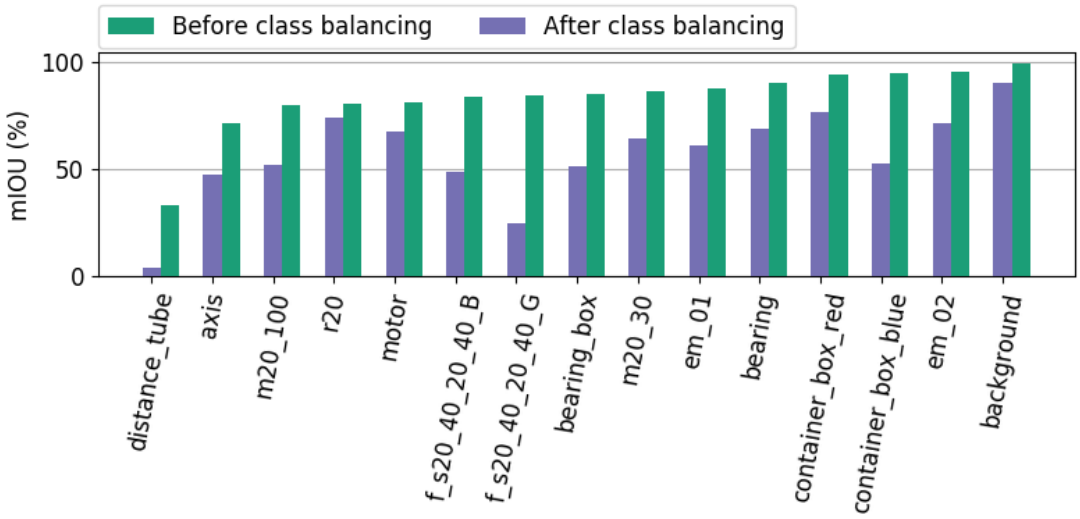
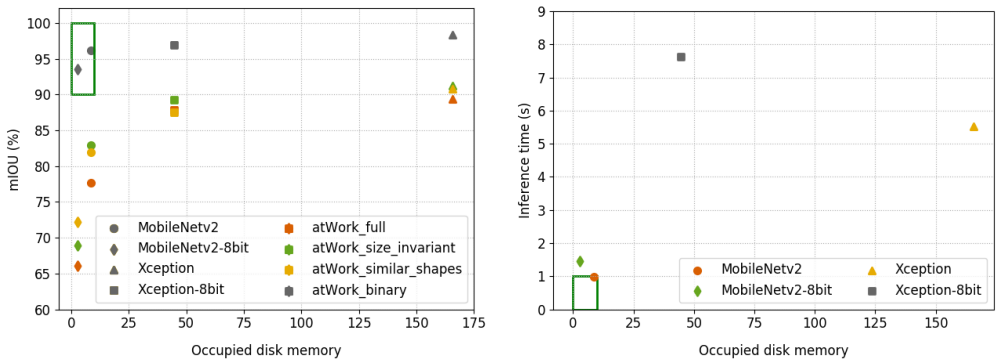


Figure 6.8: This plot shows a comparison of mIOUs obtained on each class before and after performing median-frequency re-weighting.

sion model occupies less disk memory than its corresponding full precision model. The 8bit DeepLabv3+ model with the MobileNetv2 network backbone occupies roughly 67 % less disk memory relative to the corresponding full precision model. However, the average drop in mIOU across all 4 variants of the variety of backgrounds dataset is 9.56 %. This suggests that the full precision MobileNetv2 network backbone has sufficient "learning capacity" and a drop in this "learning capacity" affects the mIOU drastically. In contrast, it can be seen that the 8bit DeepLabv3+ model with the Xception network backbone leads to roughly 73 % percent drop in memory but only 2.02 % average drop in mIOU across all variety of backgrounds dataset variants. This suggests that the low precision DeepLabv3+ with Xception network backbone still retains sufficient "learning capacity" in order to treat the low precision calculations as noise. Table 6.4 shows the mIOUs obtained.

From Table 6.5, it is evident that the number of floating operations (FLOPS) drops by rougly 94 % and 98% when quantizing DeepLabv3+ with MobileNetv2 and Xception backbones respectively. Yet, the inference time for the quantized models is more than the corresponding high precision models. Quantized models are expected to be faster as they perform low precision



(a) (b)

Figure 6.9: (a) The plot shows mIOU and disk memory occupied by both the network backbones of DeepLabv3+ on all the variants of the variety of backgrounds of dataset. Different shapes denote the different network backbones of DeepLabv3+ and different colors denote the different dataset variants. The green rectangle represents the desired region in which the mIOU is in the range [90,100] and occupied disk memory is in the range (0,10]. (b) This plot shows the inference time and disk memory occupied by full precision and quantized DeepLabv3+ models. The region indicated by a green rectangle is the desired region where the inference time is in the range (0, 1] and occupied disk memory is in the range (0, 10].

calculations. However, the inference time has evidently increased. An initial analysis seems to state that the quantized operations are not yet optimized for CPU inference based on the issue "Slow quantized graph" raised on the tensorflow source repository [1].

6.9 Discussions

This chapter addresses the different experiments performed to evaluate the DeepLabv3+ segmentation model with experiments centering around the created dataset. A summary of the observations is presented below:

- We showed that when similar objects in the dataset are combined to a single class, DeepLabv3+ achieves a better mIOU. As an extension, when all objects are combined to a single class and DeepLabv3+ is tasked with only foreground and background segmentation, the mIOU achieved is the highest.

[1] `https://github.com/tensorflow/tensorflow/issues/2807`

Dataset variant	mIOU in %			
	MobileNetv2	**MobileNetv2-8**	**Xception**	**Xception-8**
atWork_full	77.69	66.08	89.41	87.86
atWork_size_ invariant	82.95	68.88	91.19	89.30
atWork_similar_ shapes	81.91	72.22	90.75	87.50
atWork_binary	96.10	93.49	98.32	96.92

Table 6.4: This table summarizes the mIOU obtained by the quantized and full precision models of both DeepLabv3+ network backbones on all 4 variants of the variety of backgrounds dataset.

Network Backbone	Inference time (s)	Number of parameters	FLOPS	Disk memory (MB)
MobileNetv2	0.9811	2.11M	6.41B	8.7
MobileNetv2-8	1.4560	2.11M	328.87M	2.8
Xception	5.5325	41.05M	126.27B	165.6
Xception-8	7.6256	41.05M	1.94B	44.7

Table 6.5: This table summarizes the inference time, number of parameters and floating point operations (FLOPS) of both the quatized and full precision network backbones of DeepLabv3+. "M" denotes million and "B" denotes billion.

- We showed that the Xception network backbone learns better than the MobileNetv2 backbone because of its higher "learning capacity".
- We showed that the real images in the validation set needs to be made more diverse by adding images with multiple objects and occlusions in order to better understand the effectiveness of the artificial images.
- We showed that the DeepLabv3+ model favors objects which dominate the dataset. Objects which occupy higher number of pixels tend to be learned better. Another intersting observation is that the existance of distint features such as alternating concentric silver, and black circular strips in object "bearing" aids DeepLabv3+ to better distinguish the object.
- We showed that objects are similar are confused with each other in the at-

Work_full dataset variant. This confusion reduces in the atWork_size_invariant and atWork_similar_shapes variants where similar objects are combined.

- We showed that the cosine restarts learning rate decay policy leads to slightly better mIOU than the poly learning rate decay policy.
- We showed that performing median-frequency re-weighting to induce class balancing in the loss function unfortunately does not prevent the model from favoring classes which are dominant in the dataset as intended. The root cause of this result is left to be analyzed in future work.
- We showed that low precision models occupy low disk space. Low precision DeepLabv3+ with MobileNetv2 network backbone shows significant drop in mIOU but DeepLabv3+ the Xception network backbone only suffers from a minor loss in mIOU. This suggests that low precision DeepLabv3+ with MobileNetv2 network backbone cannot afford to lose any more "learning capcity" while the low precision DeepLabv3+ with Xception network backbone still retains sufficient "learning capacity" to treat low precision calculations as noise.

7

Conclusions

In this work, state-of-the-art deep learning methods for semantic segmentation was reviewed. One particular model called DeepLabv3+ with MobileNetv2 and Xception network backbones was selected. A dataset consisting of 18 atWork objects was created. This dataset also consists of artificial images generated using a artificial image generation algorithm to augment the dataset. Two different datasets called variety of backgrounds dataset and white backgrounds dataset was created which differ in terms of the background images used for the artificial image generation. Four different variants for each of the two datasets were created based on the properties of the objects in the dataset. The DeepLabv3+ model was trained on the created dataset and its performance was assessed in terms of mIOU. A quantized version of the DeepLabv3+ model was then created and was compared with the full precision DeepLabv3+ models.

7.1 Contributions

The major contributions of this work includes:

- The creation of a semantic segmentation dataset consisting of 18 atWork objects.
- Augmenting this dataset with artificial images created using an artificial image generation algorithm.

- Training DeepLabv3+ models with resource efficient network backbones and analyzing its performance.

7.2 Lessons learned

Lessons learned out of this work are listed below:

- Creating a dataset for a machine learning task is often time consuming. Capturing all the images at one stretch often introduces a bias in the dataset as most images end of being taken under the same real world conditions. A best practice would be to progressively create a dataset over a period of many days. During this time, insights from a trained model could be gathered to further improve the variety of the next round of images taken.

- Useful insights could be obtained by looking into the available data in different perspectives such as how two objects are different and what would happen if they are treated the same. Semantic segmentation models with the same hyperparameter settings could be trained on these different perspectives of data which could help understand how the model behaves in response to data.

7.3 Future work

The dataset created using the atWork objects at present consist of real images which only have one object per image. Further real images which have multiple objects in them could be added to the dataset. This would enable better assessement of the effectiveness of the artificial images in terms of aiding generalization. The trained DeepLabv3+ models currently available as a result of this work could in theory reduce the labeling cost of the additional real images. Also, additional augmentation methods such as random brightness, contrast shifts, random rotations, and others could be incorporated with the artificial image generation algorithm to improve the robustness of the generated artificial images. In addition, the use 3D CAD models of the objects in the dataset available in [1] and [2] to further improve the robustness of the generated artificial images could be considered.

The dataset created is seemingly biased towards larger objects. This bias becomes worse in the atWork_size_invariant and atWork_similar_shapes variants

in which some large objects are combined. This bias of the dataset affects the learning process and evidently, the learned DeepLabv3+ models were shown to ignore small objects. An attempt made to modify the loss function to induce class balancing has failed. This failure could be further analyzed. As an alternative approach, reducing the number of real images of large objects in comparison to smaller objects could be considered. For instance, only 2 images of containers and 22 images of "distance_tube" could be used for the training set instead of the 22 images of each.

The experiments were centered on the created dataset and did not consider hyperparameter tuning to improve results. Further tuning the different hyperparameters involved such as the decoder output stride, regularization parameter (L2 weight decay) and so on could be explored.

The initial goal of this work also included the use of pruning technqiues to compress the trained models. Pruning was not sufficiently explored in order to be implemented. The compressed DeepLabv3+ model with the MobileNetv2 network backbone using quatization suffered a high loss in mIOU of around 9 %. This is due to the naiveness involved in directly quantizing for the sake of compression. A better approach would be to first prune redundant weights, filters or feature maps and later perform quantization which is reported to be effective in the literature. Recently, reinforcement learning has been applied to train an agent to prune a CNN model [20]. This approach could also be explored.

A

Further details regarding the Dataset

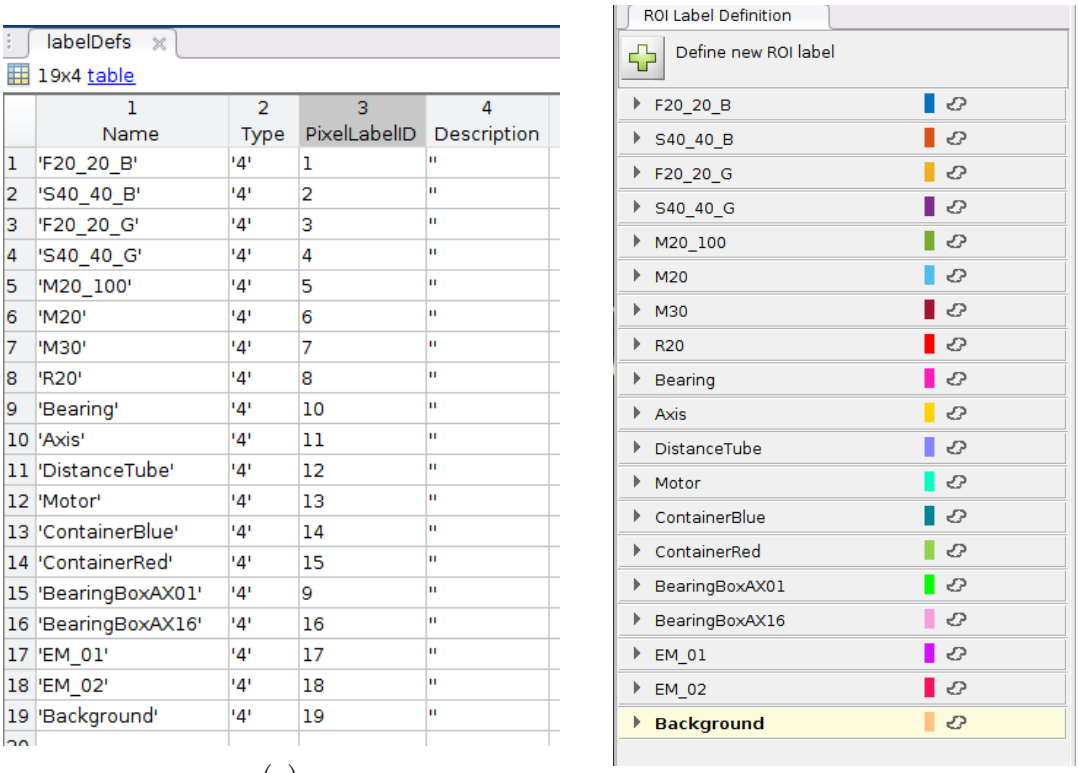
A.1 Selection of a labeling tool

[Incomplete]

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.



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	''

(a)

(b)

Figure A.1: (a) Contents of the labelDefs .mat file, (b) ROI Label Definitions window.

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

A.2 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 Figure A.1a.

The ImageLabeler app, by default, provides different tools which help create pixelwise labels. The tools are shown in the Figure A.2. These tools become accessible once an image and the label definitions are loaded. A short description

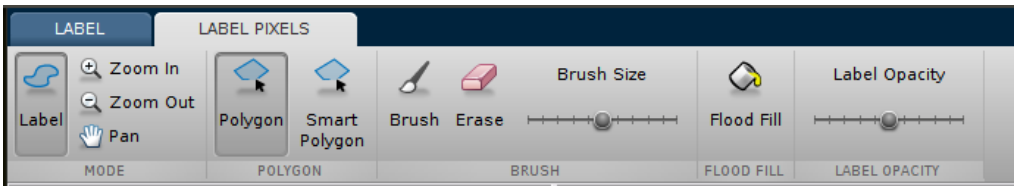


Figure A.2: Tools provided by the ImageLabeler app

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.
- 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 window shown in Figure A.1b. An example of an object in the ImageLabeler tool once the annotation is complete is shown in Figure A.3.

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:

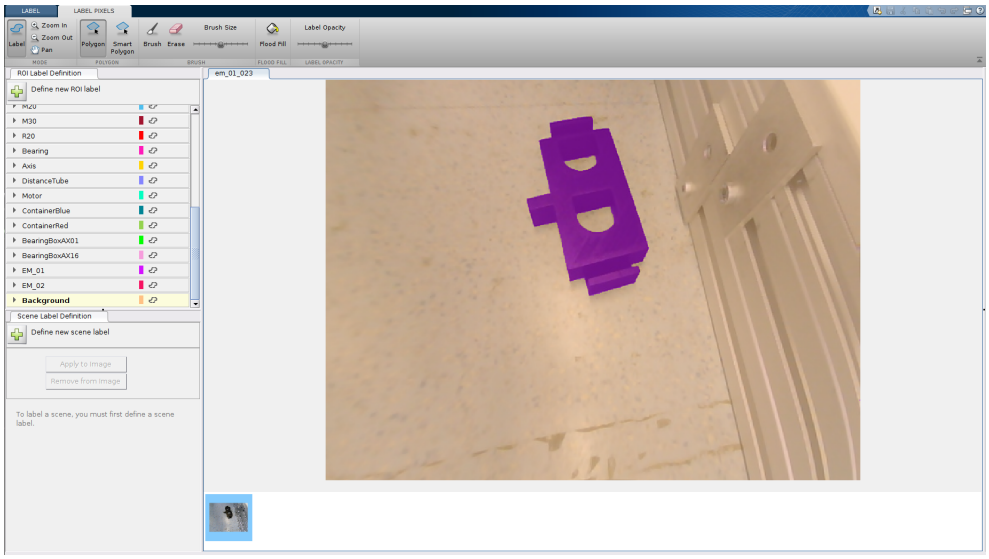
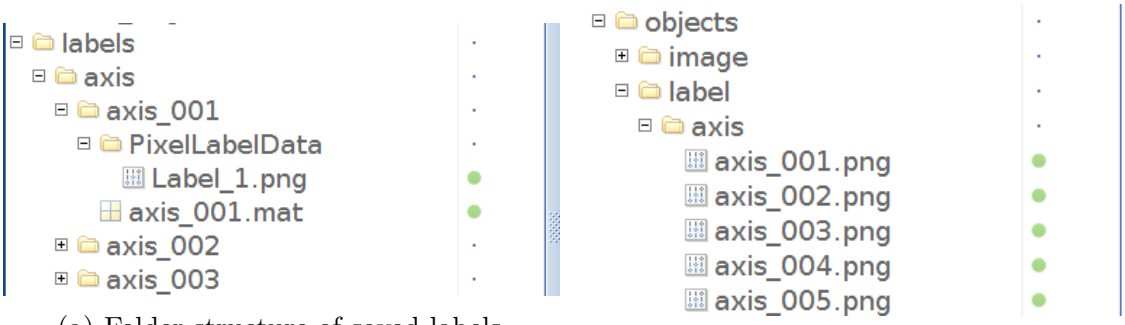


Figure A.3: An object labeled in the ImageLabeler.

- 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 Figure A.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 seen in Figure A.4a) is the label image. This image is renamed to have the same name as the image file and a folder structure as in Figure A.4b is created by using a python code.



(a) Folder structure of saved labels

(b) Rearranged folder structure

Figure A.4: Different folder structures

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

A.3 Search keywords for background images

Keywords used for downloading background images are listed in Table A.1.

A.4 Generator option details

Further details regarding the generator options used to configure the artificial image generation algorithm is given in Table A.2.

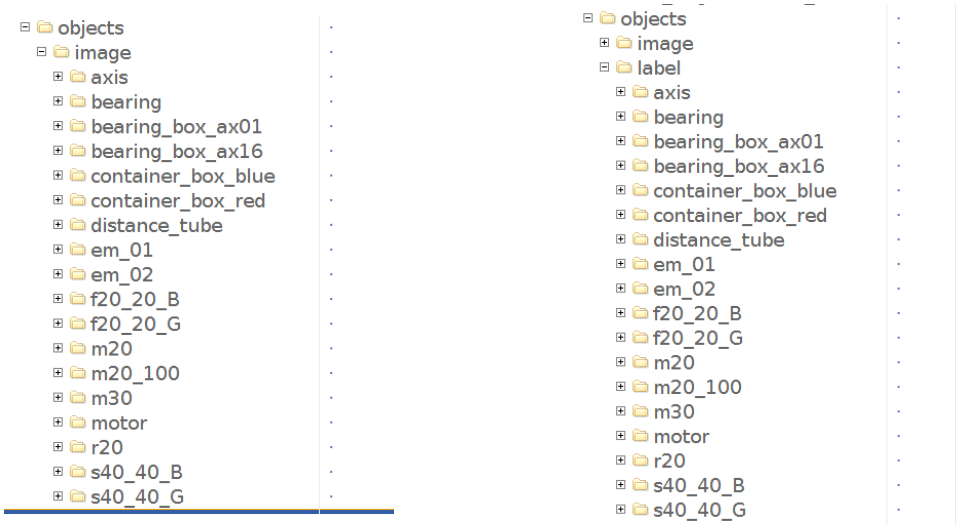


Figure A.5: Folder structure showing different object folders in both image and label folders.

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 A.1: This table lists the keywords used to download images used as background for artificial image generation.

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

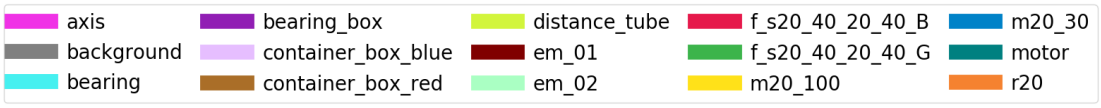
Table A.2: Default value of generator options and whether the options are required to be set.

B

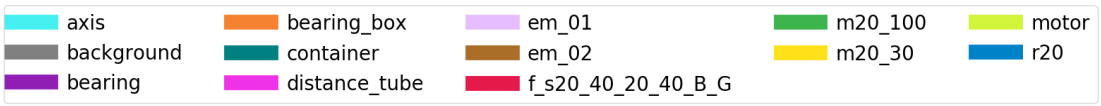
Sample predictions

In this appendix chapter, visualizations of the predictions obtained through DeepLabv3+ models is provided. In Figure B.1, the color used for visualizing each objects in all 4 dataset variants are shown. In Figures B.2, B.3, B.4 and B.5, the first column shows the input images, the second column shows the corresponding ground truths, the third column shows the predictions obtained with MobileNetv2 network backbone and the fourth column shows the predictions obtained with the Xception network backbone. Also, in Figures B.2, B.3, B.4 and B.5, results shown in the first two rows are from DeepLabv3+ trained on the variety of backgronds dataset and the results in the third row is from DeepLabv3+ trained on the white backgrounds dataset.

(a) atWork_full



(b) atWork_size_invariant



(c) atWork_similar_shapes

(d) atWork_binary

Figure B.1: Object names and corresponding colors used for the visualiztion of different dataset variants.

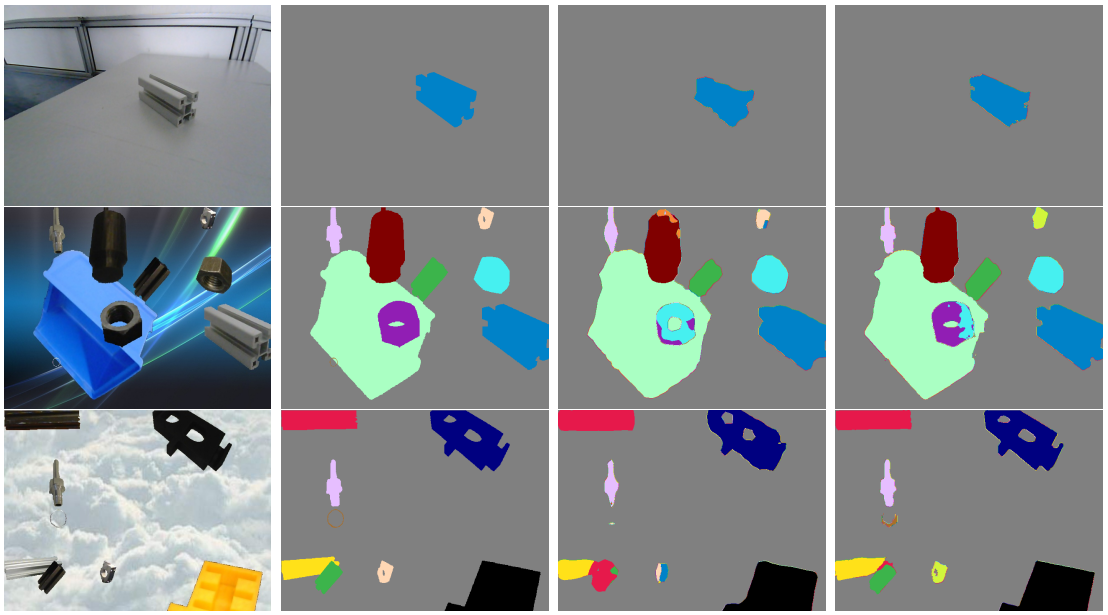


Figure B.2: Predictions of DeepLabv3+ on the **atWork_full** variant of the variety of backgrounds dataset and the white backgrounds dataset.

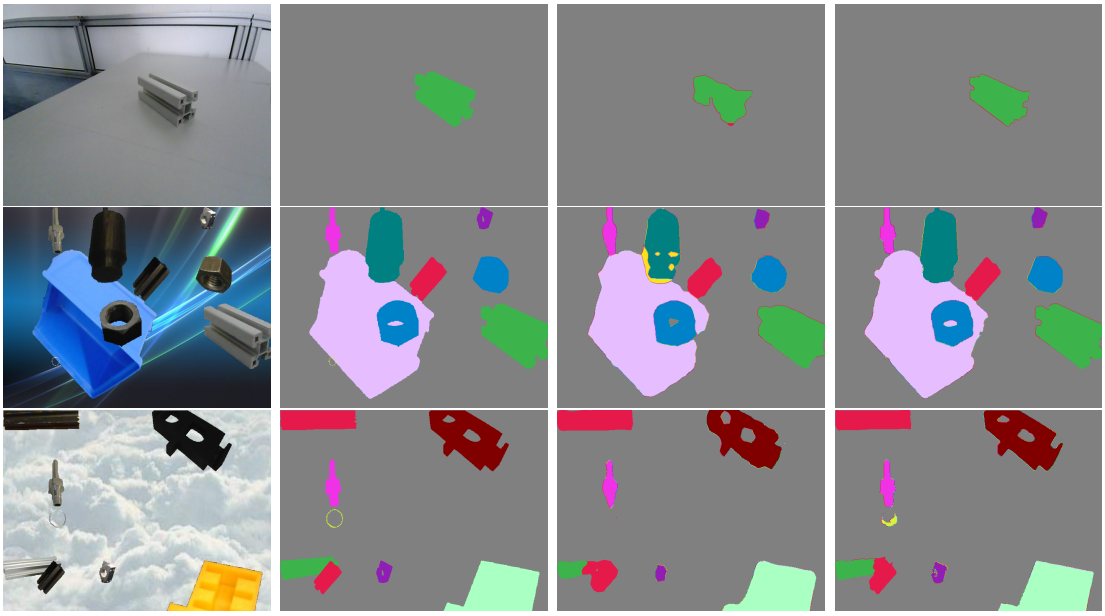


Figure B.3: Predictions of DeepLabv3+ on the **atWork_size_invariant** variant of the variety of backgrounds dataset and the white backgrounds dataset.

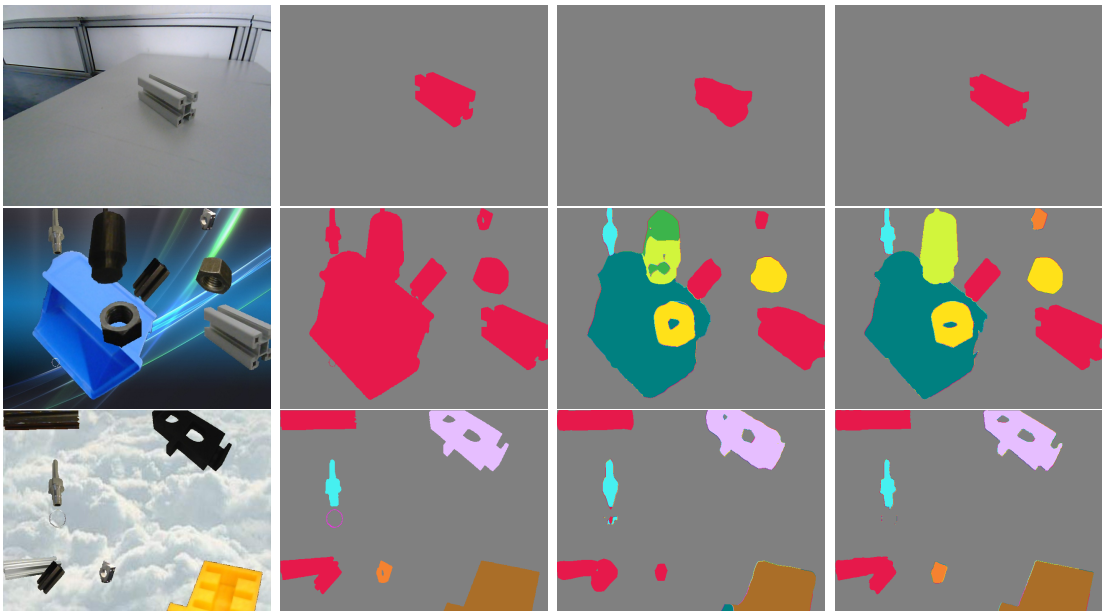


Figure B.4: Predictions of DeepLabv3+ on the **atWork_similar_shapes** variant of the variety of backgrounds dataset and the white backgrounds dataset.

Figure B.5: Predictions of DeepLabv3+ on the **atWork_binary** variant of the variety of backgrounds dataset and the white backgrounds dataset.

C

Hyperparameters

Your second chapter appendix

References

[1] 3d cad models of the manipulation objects, platforms and other environmental items, . URL https://github.com/robocup-at-work/models. Online accessed: 2018-08-11.

[2] This repository contains all kind of 2d/3d models of the rockin@work test bed environment, . URL https://github.com/rockin-robot-challenge/at_work_models. Online accessed: 2018-08-11.

[3] Cs231n: Convolutional neural networks for visual recognition. URL http://cs231n.github.io/convolutional-networks/. Online accessed: 2018-08-08.

[4] Convolutional neural network. URL https://de.mathworks.com/solutions/deep-learning/convolutional-neural-network.html. Online accessed: 2018-08-08.

[5] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *CoRR*, abs/1511.00561, 2015. URL http://arxiv.org/abs/1511.00561.

[6] Eli Bendersky. Depthwise separable convolutions for machine learning. URL https://eli.thegreenplace.net/2018/depthwise-separable-convolutions-for-machine-learning/. Online accessed: 2018-08-03.

[7] Jan Carstensen, Nico Hochgeschwender, Gerhard Kraetzschmar, Walter Nowak, and Sebastian Zug. Robocup@work rulebook version 2016. URL http://www.robocupatwork.org/download/rulebook-2016-01-15.pdf. Online accessed: 2018-08-03.

[8] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *CoRR*, abs/1412.7062, 2014. URL `http://arxiv.org/abs/1412.7062`.

[9] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *CoRR*, abs/1606.00915, 2016. URL `http://arxiv.org/abs/1606.00915`.

[10] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *CoRR*, abs/1706.05587, 2017. URL `http://arxiv.org/abs/1706.05587`.

[11] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. *CoRR*, abs/1802.02611, 2018. URL `http://arxiv.org/abs/1802.02611`.

[12] François Chollet. Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357, 2016. URL `http://arxiv.org/abs/1610.02357`.

[13] Daphne Cornelisse. An intuitive guide to convolutional neural networks. URL `https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050`. Online accessed: 2018-08-09.

[14] Cityscapes dataset. Examples of fine annotations. URL `https://www.cityscapes-dataset.com/examples/#fine-annotations`. Online accessed: 2018-08-03.

[15] Arden Dertat. Applied deep learning - part 4: Convolutional neural networks. URL `https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2`. Online accessed: 2018-08-09.

[16] First350. Tensorflowdealing with imbalanced data. URL https://blog.node.us.com/tensorflow-dealing-with-imbalanced-data-eb0108b10701. Online accessed: 2018-08-04.

[17] Alberto Garcia-Garcia, Sergio Orts-Escolano, Sergiu Oprea, Victor Villena-Martinez, and José García Rodríguez. A review on deep learning techniques applied to semantic segmentation. *CoRR*, abs/1704.06857, 2017. URL http://arxiv.org/abs/1704.06857.

[18] Simon Haykin. *Neural Networks and Learning Machines, Third Edition*. URL http://dai.fmph.uniba.sk/courses/NN/haykin.neural-networks.3ed.2009.pdf. Online accessed: 2018-08-08.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

[20] Qiangui Huang, Shaohua Kevin Zhou, Suya You, and Ulrich Neumann. Learning to prune filters in convolutional neural networks. *CoRR*, abs/1801.07365, 2018. URL http://arxiv.org/abs/1801.07365.

[21] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440, June 2015. doi: 10.1109/CVPR.2015.7298965.

[22] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1411.4038, 2014. URL http://arxiv.org/abs/1411.4038.

[23] Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with restarts. *CoRR*, abs/1608.03983, 2016. URL http://arxiv.org/abs/1608.03983.

[24] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient transfer learning. *CoRR*, abs/1611.06440, 2016. URL http://arxiv.org/abs/1611.06440.

[25] Adam Paszke, Abhishek Chaurasia, Sangpil Kim, and Eugenio Culurciello. Enet: A deep neural network architecture for real-time semantic segmentation. *CoRR*, abs/1606.02147, 2016. URL http://arxiv.org/abs/1606.02147.

[26] Sarah Perez. Google makes the camera smarter with a google lens update, integration with street view. URL https://techcrunch.com/2018/05/08/google-makes-the-camera-smarter-with-a-google-lens-update-integration-with-?guccounter=1. Online accessed: 2018-08-03.

[27] Paul-Louis Prve. An introduction to different types of convolutions in deep learning. URL https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d. Online accessed: 2018-08-10.

[28] 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.

[29] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. *CoRR*, abs/1802.01548, 2018. URL http://arxiv.org/abs/1802.01548.

[30] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015. URL http://arxiv.org/abs/1506.02640.

[31] 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.

[32] Mark Sandler and Andrew Howard. Mobilenetv2: The next generation of on-device computer vision networks. URL https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html. Online accessed: 2018-08-03.

[33] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks

for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018. URL http://arxiv.org/abs/1801.04381.

[34] Nathan Silberman. Nyu depth dataset v2. URL https://cs.nyu.edu/~silberman/datasets/nyu_depth_v2.html. Online accessed: 2018-08-03.

[35] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. *CoRR*, abs/1406.2199, 2014. URL http://arxiv.org/abs/1406.2199.

[36] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. URL http://arxiv.org/abs/1409.4842.

[37] Sasha Trubetskoy. List of 20 simple, distinct colors. URL https://sashat.me/2017/01/11/list-of-20-simple-distinct-colors/. Online accessed: 2018-08-03.

[38] Hardik Vasa. hardikvasa/google-images-download. URL https://github.com/hardikvasa/google-images-download/. Online accessed: 2018-08-03.

[39] Pete Warden. How to quantize neural networks with tensorflow. URL https://petewarden.com/2016/05/03/how-to-quantize-neural-networks-with-tensorflow/. Online accessed: 2018-08-03.

[40] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. *CoRR*, abs/1512.06473, 2015. URL http://arxiv.org/abs/1512.06473.

[41] Hengshuang Zhao, Xiaojuan Qi, Xiaoyong Shen, Jianping Shi, and Jiaya Jia. Icnet for real-time semantic segmentation on high-resolution images. *CoRR*, abs/1704.08545, 2017. URL http://arxiv.org/abs/1704.08545.

[42] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. *CoRR*, abs/1611.01578, 2016. URL http://arxiv.org/abs/1611.01578.