FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

Sercan Turkmen

Scene Understanding Through Semantic Image Segmentation in Augmented Reality

Master's Thesis
Degree Programme in Computer Science and Engineering
June 2019

Turkmen S. (2019) Scene Understanding Through Semantic Image Segmentation in Augmented Reality. University of Oulu, Degree Programme in Computer Science and Engineering, 64 p.

ABSTRACT

Semantic image segmentation, the task of assigning a label to each pixel in an image, is a major challenge in the field of computer vision. Semantic image segmentation using fully convolutional neural networks (FCNNs) offers an online solution to the scene understanding while having a simple training procedure and fast inference speed if designed efficiently. The semantic information provided by the semantic segmentation is a detailed understanding of the current context and this scene understanding is vital for scene modification in augmented reality (AR), especially if one aims to perform destructive scene augmentation. Augmented reality systems, by nature, aim to have a real-time modification of the context through head-mounted see-through or video-see-through displays, thus require efficiency in each step. Although there are many solutions to the semantic image segmentation in the literature such as DeeplabV3+ [1], Deeplab DPC [2], they fail to offer a low latency inference due to their complex architectures in aim to acquire the best accuracy. As a part of this thesis work, we provide an efficient architecture for semantic image segmentation using an FCNN model and achieve real-time performance on smartphones at 19.65 frames per second (fps) while maintaining a high mean intersection over union (mIOU) of 67.7% on Cityscapes validation set with our "Basic" variant and 15.41 fps and 70.3% mIOU on Cityscapes test set using our "DPC" variant. The implementation is open-sourced and compatible with Tensorflow Lite, thus able to run on embedded and mobile devices.

Furthermore, the thesis work demonstrates an augmented reality implementation where semantic segmentation masks are tracked online in a 3D environment using Google ARCore [3]. We show that the frequent calculation of semantic information is not necessary and by tracking the calculated semantic information in 3D space using inertial-visual odometry that is provided by the ARCore framework, we can achieve savings on battery and CPU usage while maintaining a high mIOU. We further demonstrate a possible use case of the system by inpainting the objects in 3D space that are found by the semantic image segmentation network. The implemented Android application performs real-time augmented reality at 30 fps while running the computationally efficient network that was proposed as a part of this thesis work in parallel.

Keywords: 3D tracking, inpainting, real-time, efficient, mobile, Android, video-see-through display, fully convolutional neural network

TABLE OF CONTENTS

FOREWORD

This master's thesis was carried out to explore scene understanding in augmented reality for mobile devices at the Center for Machine Vision and Signal Analysis (CMVS), University of Oulu. First and foremost, I would like to express my gratitude to Prof. Janne Heikkilä for his supervision and guidance throughout my master's thesis and studies. In addition, I would like to extend my thanks to Dr. Matteo Pedone for his review and feedback. I am also deeply grateful to my family for being continuing source guidance, encouragement, and support. Last but not least, I would like to thank to my friends for their support and Rin Motomura who has always been an unfailing source of encouragement, support, and reassurance.

Oulu, 18th June, 2019

Sercan Turkmen

LIST OF ABBREVIATIONS AND SYMBOLS

AR	Augmented Reality
VR	Virtual Reality
AV	Augmented Virtuality
MR	Mixed Reality
FCNN	Fully Convolutional Neural Network
IOU	Intersection Over Union
mIOU	Mean Intersection Over Union
COCO	Microsoft Common Objects in Context
ROI	Region of Interest
ML	Machine Learning
SLAM	Simultaneous Localization and Mapping
CPU	Central processing unit
GPU	Graphics processing unit
SCIA	Scandinavian Conference on Image Analysis
ILSVRC	ImageNet Large Scale Visual Recognition Competition
VOC	Visual Object Classes
API	Application programming interface
FLOP	Floating Point Operation
MP	Megapixel
RMSE	Root Mean Squared Error
STD	Standard deviation
GLES	OpenGL for Embedded Systems
MAC	Memory Access Cost
ReLU	Rectified Linear Unit

1. INTRODUCTION

Augmented reality is an important topic of the field of computer vision. In recent years, there has been a renewed interest in augmented reality (AR) since its first introduction as virtual fixtures [4] in 1992. As a variation of virtual reality (VR), the concept of putting the user in a completely virtual environment, AR aims user to see the real world while it is superimposed upon or composited with virtual objects [5]. By doing so, AR has shown to improve the agent performance and understanding by offering supplemental information to the real-world in [4]. Furthermore, the advancements on the efficient algorithms and faster processors, AR was able to be demonstrated on mobile phones thus allowing easy access to the technology. Although currently, the main application of AR on mobile is in the video gaming industry, AR can be applied in numerous fields such as archaeology [6], architecture [7], visual art [8], commerce [9], education [10] and so on. Such an extensive range of applications shows that AR is an important topic.

AR technology can be used to augment any of the five senses of a user by overlaying images, audio, video, and touch or haptic sensations [8]. In the case of visual sense, the act of sensory information augmentation of the environment can be constructive (i.e. adding virtual objects to the real world) or destructive (i.e. masking the parts of the real-world environment) [4]. Examples of constructive visual augmentation widely exist in the current literature such as [6, 7, 8, 9, 10], and in everyday mobile applications such as Houzz [11] and IKEA Place [12]. On the other hand, destructive augmentation often requires an understanding of the current scene. An example of destructive would be darkening of the unrelated areas for the completion of the task to keep higher attention on the task itself.

To achieve destructive augmentation, the environment should have a model in which we can understand what to overlay or where to mask. If no prior modeling of the setting exists, then, an online modeling process is needed to map the target pixels of the visible scene. A way to get the scene understanding of the current context without any prior information is to run classification methods such as object detection. Using object detection one can understand the current objects which are visible in the current context. But the limitation of object detection is that the regions are not masked, meaning that only the bounding boxes of the objects are selected, thus leaving uncertainty on the exact pixels of the object within the found region of interest (ROI). To tackle this challenge, a method that is capable of doing classification on a pixel basis is required.

Semantic image segmentation, as a way to get an advanced understanding of the current environment, makes it possible to perform destructive augmentation at pixel level accuracy. For example, someone who will give a speech to a crowd might want to remove the audience from his environment to feel less stressed. Using semantic segmentation to detect the pixels that belong to the human crowd and applying destructive augmentation would be the desire of the user.

This raises the question of what is semantic image segmentation and how can it be used in such scenarios? Semantic image segmentation is the task of pixel level labeling of each object or region in an image and is a major challenge in computer vision [13]. The recent studies show that fully convolutional networks (FCNNs) are the state-of-art solution to semantic segmentation [13]. Furthermore, such networks

offer simplicity and speed during the learning and prediction process. Semantic image segmentation has many applications such as autonomous driving, robotic navigation, and localization. These models are trained by supervised learning using numerous images that have labels for each pixel. Over the recent years, many data sets that offer annotations at pixel level have been emerged such as ADE20K [14], COCO [15], Cityscapes [16], PASCAL VOC [17], thus boosting the scene understanding in the field.

The task of semantic image segmentation using FCNNs are computationally heavy tasks and would require proper designs to run predictions at real time on everyday items such as mobile phones. Recently, there has been an increasing interest in the field of computer vision on computationally efficient approaches such as [18, 19, 20, 21]. By exploiting these approaches, it is now possible to efficiently run neural network inference on mobile devices in a timely manner.

1.1. Aim of the thesis

This thesis intends to explore destructive visual augmentation on mobile devices by using semantic image segmentation for the automatic understanding of the scene and determination of the regions of interests without a need for prior modeling of the setting. Later, the found object masks are tracked in the 3D environment by exploiting the current state-of-art plane and environment tracking offered by ARCore [3]. In its core, this thesis work aims to perform online and real-time destructive scene augmentation by performing online scene understanding, masking, and tracking of objects in a 3D environment on a mobile phone.

1.2. Structure of the thesis

The overall structure of the thesis takes the form of seven distinct chapters. This chapter, as the first chapter, gives an introduction to the thesis work by defining some of the concepts briefly and stating the aim of the thesis. The second chapter gives a brief background to semantic image segmentation and explains the state-of-art in the field by focusing on learning-based solutions. The third chapter examines the augmented reality by a historical overview and defining the current challenges and findings. After that, Chapter 4 describes the implementation details of semantic segmentation method used in the thesis work and the mobile application. Then, Chapter 5 presents the evaluation method and the results of the evaluation of the thesis work. Afterward, a discussion where we discuss the implication of the findings to future research into destructive vision augmentation is presented. Finally, we conclude the thesis work in Chapter 7.

2. SEMANTIC IMAGE SEGMENTATION

The purpose of this chapter is to review the literature on semantic image segmentation. It begins by describing the problem and follows by covering specific challenges that are currently in the field of computer vision. After that, an overview of learning based solutions to the problem is given. Since this thesis focuses on online scene understanding, this chapter is crucial for the work that has been done for the final implementation. That is why this chapter describes learning-based semantic segmentation solution in great detail, going over many of the layers that have been used in this work.

2.1. Problem Formulation

An image in a general sense is all types of media that can be understood visually by humans such as video, animation, a still image, graphics and so on. From these images, we, as human beings, gather information about the real world. Many techniques for understanding these images to acquire more information have been developed and applications have been brought to existence. These techniques can be categorized under a three-layer generic framework, so-called image engineering. As can be seen in Figure 1, these layers are image processing (low layer), image analysis (middle layer) and image understanding (high layer). [22]

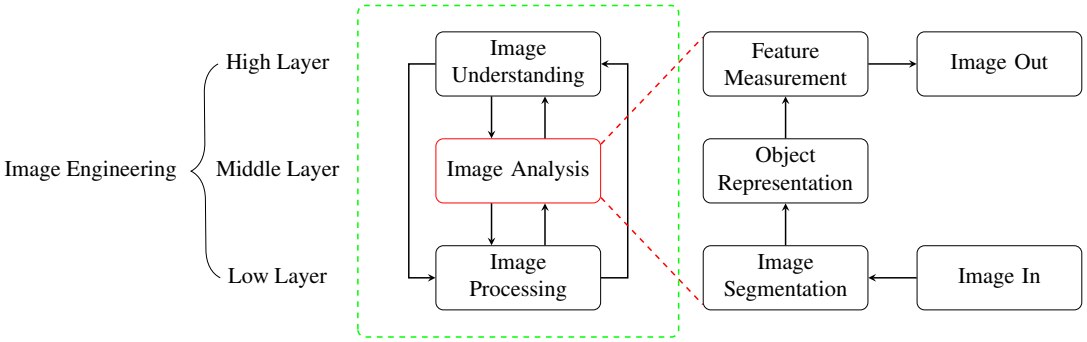


Figure 1. Image engineering framework

Image segmentation as it is one of the most important steps in the tasks of image analysis due to its effect of the other steps in the framework. Segmentation has a significant effect on the accuracy of the object representation and feature measurement steps. [22]

In the following, we will talk about image segmentation in detail. Image segmentation is the term that refers to partitioning an image into a set of patches that cover it. A common aim of image segmentation is to identify the meaningful regions of an image such as the vegetation, urban areas or crops of a satellite image. It is still valid to do segmentation even if the regions do not cover the entire image, in that case, regions can be separated into foreground regions of interest and background regions that are ignored. [23]

There are two main objectives of segmentation. First, decomposition of the image into parts for other analysis. As a simple case example, if the colors of the environment

are well controlled (areas of interest are not similar colors with the background) human face can be extracted from the background on a color image using the color components of a human face. But in complex cases, such as finding the road network from a grey scale aerial image, the segmentation task becomes very difficult and may require domain building knowledge. Second, the objective can be to perform a change of representation where pixels are categorized into higher-level units to be either more meaningful or efficient for further analysis. [23]

Segmentation is one of the oldest challenges in the area of computer vision. The first instance of image segmentation can be considered as the Roberts operator [24] which is an operator for detecting edges among various parts of an image that was introduced in 1965. It was the first breakthrough of decomposing an image into its sub-part regions. After that, many methodologies and algorithms for image segmentation have been developed. The extent of media is also have been extended to cover not just 2D images but also 3D images, videos, multi-band images and so on. [22]

Despite over half a decade of research since its first development in 1965, image segmentation is still an open issue. It is pointed out in [22] that the number of research on image segmentation is steadily increasing. Furthermore, it becomes necessary to identify the groups of algorithms for the task. In [22], these methodologies are grouped as follows:

1. Thresholding,
2. Pixel classification which includes relaxation, Markov random field based approaches, and neural network based approaches,
3. Range image segmentation,
4. Color image segmentation,
5. Edge detection,
6. Methods based on the fuzzy set theory which includes fuzzy thresholding, fuzzy clustering, and fuzzy edge detection.

As a part of this thesis work, we will be focusing on the Item 2 where we aim to classify each pixel in the image using a neural network based approach. This is the so-called semantic image segmentation which is a major challenge in computer vision and defined as assigning a label to every pixel in an image [1, 15]. Fully convolutional networks (FCNNs) are proved to be the state-of-art solution to this task over the recent years while providing speed during training and inference [13]. Such architectures have a broad range of applications currently on the field such as autonomous driving, robotic navigation, localization, and scene understanding. This thesis work exploits scene understanding using an FCNN based solution to the task of semantic image segmentation.

Instance awareness One important concept to realize is the instance-awareness of a semantic segmentation method. When we think about an ordinary image, it may contain several instances of the same category such as four giraffes as shown in Figure 2. In this extent, an algorithm is considered to be instance-aware if each instance of

the category is segmented separately. In this thesis work, the aim is to find the pixels of specific categories rather than spotting the different instances. Thus, the solution presented for scene understanding in the context of this thesis is not an instance-aware solution.

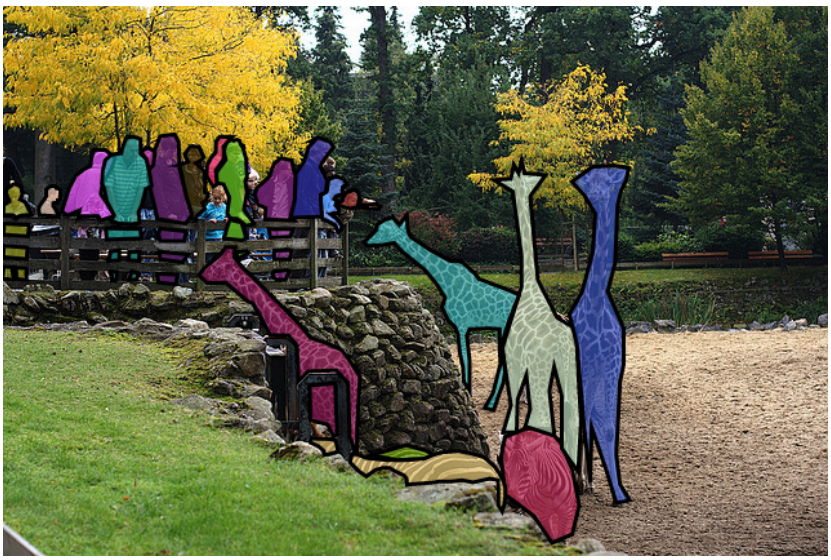


Figure 2. A semantic image segmentation example[1].

2.2. Learning Based Approaches

This section describes the learning based approaches to the semantic image segmentation task. The section starts with an introduction to machine learning and deep learning and then covers fully convolutional neural networks and some specific network layers that are used in these architectures. This thesis work is heavily based upon FCNN concepts and thus it is necessary to have a detailed overview of the topic.

Conventional machine learning methods require a type of feature representation since their ability of processing raw data, for example, every pixel in an image, is limited. To construct machine learning or pattern recognition systems one often needed a lot of domain expertise to compose a feature extractor which transforms the raw data–e.g. pixel values of an image– into a usable internal format or a feature vector. Then, such representations would be used to detect or classify patterns in the input by a learning subsystem. [25]

Representation Learning The representations which are needed for the classifiers or detectors can be automatically discovered from the raw data using representation learning. Deep learning methods are representation learning approaches which have multiple layers where each layer consist of simple but non-linear modules to obtain the representation from the raw data. At each layer, the representations become more

[1]Image by Microsoft COCO website licensed under `https://creativecommons.org/licenses/by-sa/4.0/deed.en` taken from `http://cocodataset.org/\#explore?id=503832`

abstract which results in the ability to learn very complex functions with enough compositions of this kind of transformations. In case of an image, the learned features at the earlier layers often amplify the edges, the middle layers may spot the particular position arrangements of these edges and the latest layers may create larger combinations of parts of recognizable objects. Combinations of these layers are then used to detect objects. But, the main topic here is that the representation functions are not designed by human engineers, instead, they are learned from the raw data using a generic learning procedure. [25]

2.2.1. Deep Neural Networks

Deep learning, as a subset of machine learning (ML), gained a huge interest since 2012 with the success of AlexNet [26] in image classification on the ILSVRC-2012 [27] challenge (known as ImageNet challenge). Furthermore, over the recent years, the deep learning approaches have been the best performing approaches in many tasks such as object detection [27, 15, 28], speech recognition [29] and so on. Semantic image segmentation as a branch of image recognition is not an exception to this success. The state-of-art solution to the current challenges in the field such as [15, 17, 16] has been proven to be FCNN based solutions such as [30, 1, 2, 13]. According to [25], deep learning has potential to be much more successful in the future as it demands very little engineering by hand. And by doing so, it can benefit from the increasing amount of computational power and data.

Such deep learning models are often trained using a supervised learning method. There are different kinds of learning techniques to train a machine learning algorithm whether it is deep learning or not. The supervised learning is one of these techniques where, first, a large set of data is collected and labeled accordingly to the categories. For example, if we would like to classify cars, people, dogs and cat, a large set of images of each class can be collected and labeled. Then, the machine has shown an image and produce a set of scores for each class. Here, our desire is to have the highest score for the correct category and to achieve this, training of the system is necessary. To train the model, we calculate an objective function to find the error between the system output and our desired set of scores. The inner adjustable parameters of the model (weights) are then tuned to for reducing the error. In the case of deep learning, there might be millions of these parameters in the system which are then adjusted over millions of labeled data.

Neurons (also known as unit or node) are the building blocks of a deep learning network. A visual representation of a single neuron is shown in Figure 3. As shown in Figure 4, each layer of a deep learning model contains several of these neurons. The neurons calculate a simple function from the input and transfer it to the output. The calculation is formulated by

$$y = \theta\Big(b + \sum_{i=1}^{N} x_i w_i\Big) \tag{1}$$

where θ is the activation function, b is the bias, N is the number of inputs, x are the inputs and w are the weights.

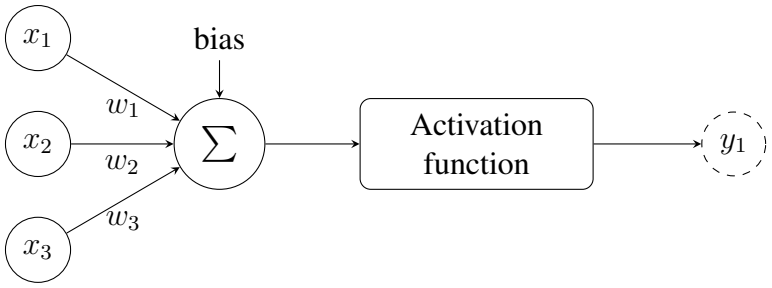


Figure 3. A single neuron.

Each layer of a deep learning model may consist of thousands of these neurons. These layers are then connected to each other to form the network architecture. To train the model, the input is given to the network and an error according to the loss function, such as [31], is calculated. The weights and the biases for each node are then updated using back-propagation to minimize the loss. A simple forward flow example of a neural network can be seen in Figure 4. Adjusting weights during the back-propagation is exemplified in Figure 5 where t_k denotes the desired value for the given input.

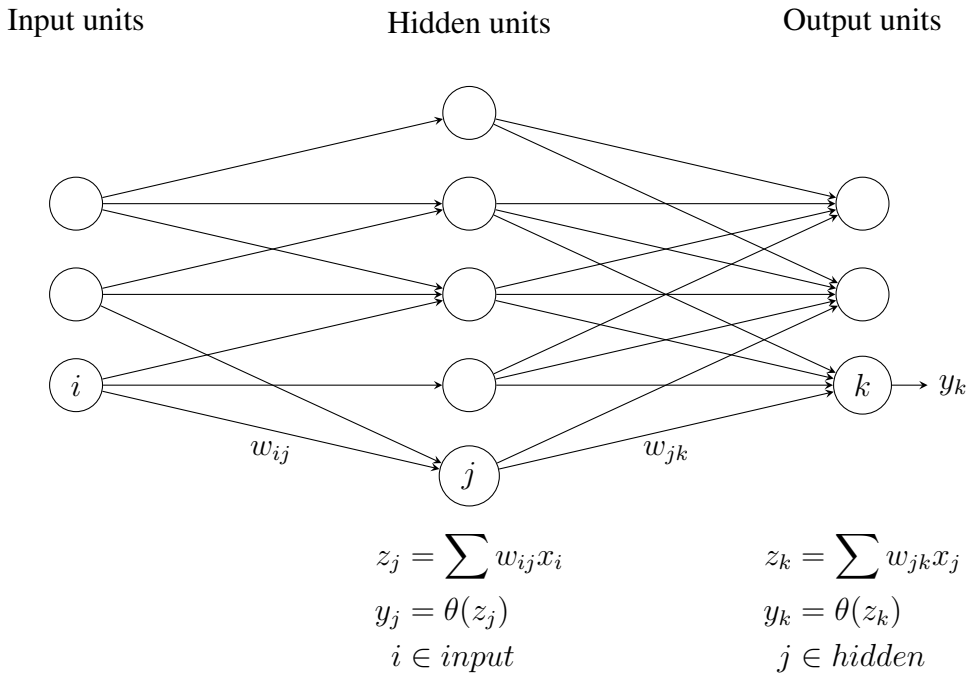


Figure 4. Example of a simple neural network forward flow without biases.

2.2.2. Fully Convolutional Neural Networks

In conventional classification task, where the aim is to predict the class of an image, an input image first goes through several convolutional layers and then fully connected layers, where all the neurons in the layer have connections to all of the previous layer's

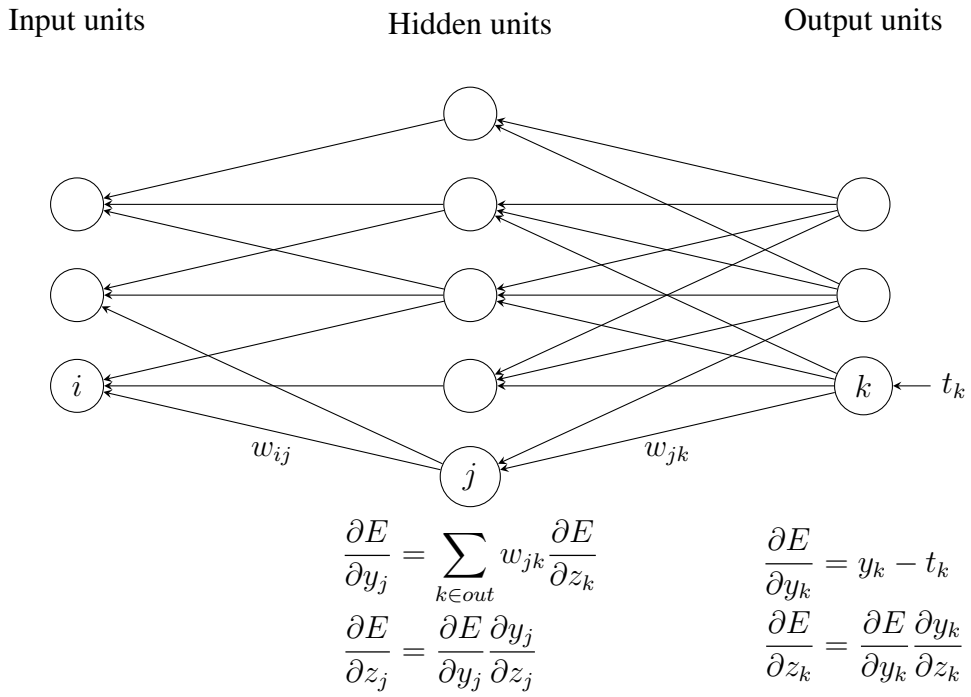


Figure 5. Example of a simple neural network back-propagation without biases. t_k denotes the desired output. The error between the desired output and generated output is calculated, and then back-propagated to determine the new weights.

neurons, which results in a vector of class probabilities as shown in Figure 6. But for the semantic segmentation task, fully convolutional networks (FCNNs), a subset of convolutional networks, are proposed and proved to be the state-of-art approach for the first time in [13]. FCNNs consists of only convolutional layers in their architectures and they are used for semantic image segmentation as shown in Figure 7. They belong to the representation learning category where the input is the raw pixel data and the output is generated from these input without any feature engineering.

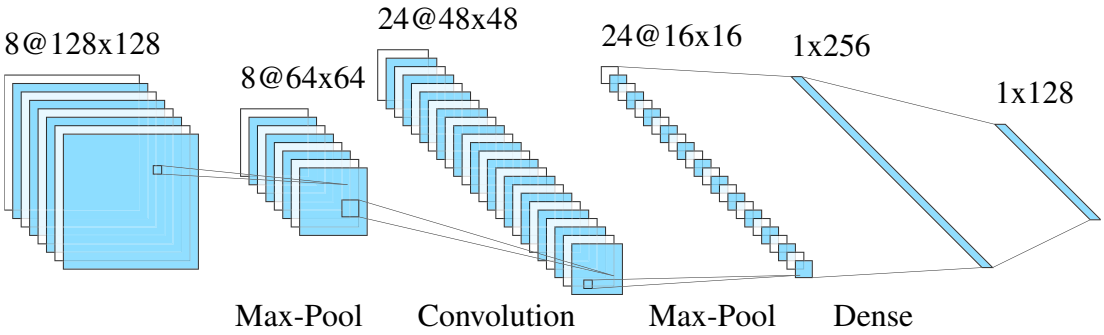


Figure 6. Example of a fully connected classification network.

Recent work in the field of semantic segmentation [31, 30, 1, 2, 32, 33, 34] follows the trend of using FCNNs. As a solution to scene understanding of the current context, this thesis work uses an FCNN based approach as it provides simplicity during training

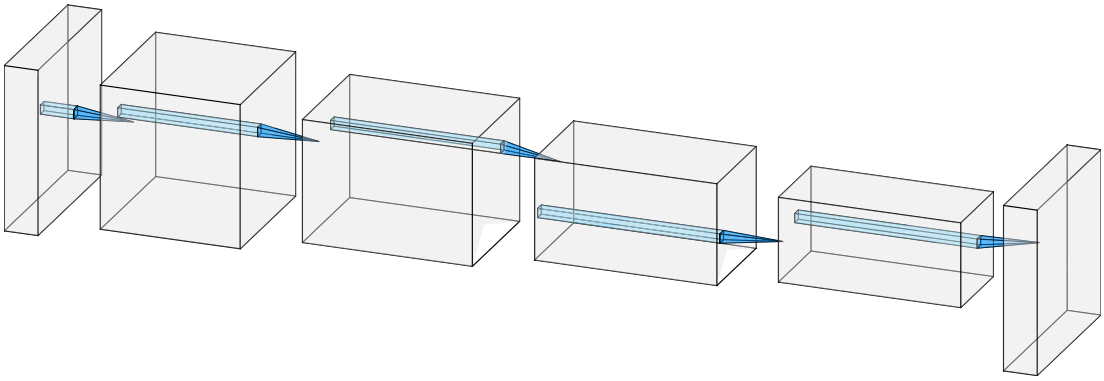


Figure 7. Example of a fully convolutional network.

and fast inference time even on mobile devices. In the following subsections, typical layers that are used in FCNNs are going to be explained in detail.

Convolutional layer

As we have discussed earlier, each layer of a neural network consists of neurons that are connected to the previous layer. The convolutional layer is a specialized layer that is used as a building block for convolutional neural networks. The neurons in a convolutional layer are organized in feature maps, where each neuron is connected to local patches of the previous feature map through a collection of weights. Result of this calculation is then evaluated by a non-linearity function such as a rectified linear unit (ReLU) [35] which is formulated as

$$f(x) = max(0, x) \tag{2}$$

and is a commonly used activation function in the literature. This set of weights are shared among all the neurons in the same feature map. There are two motivations to this layer. [25]

1. Array-based data, such as an image, often contain highly correlated local groups of values which can model easy to detect distinctive local motifs. For example, edges of an object can be seen in a local patch of an image.

2. A motif can appear anywhere in an image, thus the reason for having the same weights for neurons at different locations. This allows detection of the same pattern in various locations of an image.

The convolutional layer is an optimization for the neural networks. It limits the connections of neurons to the local patches and weight sharing reduces the memory complexity. In [36], it is calculated that a convolutional layer is more efficient than a matrix multiplication about roughly 60 000 times. An example illustration of a convolutional layer is shown in Figure 8.

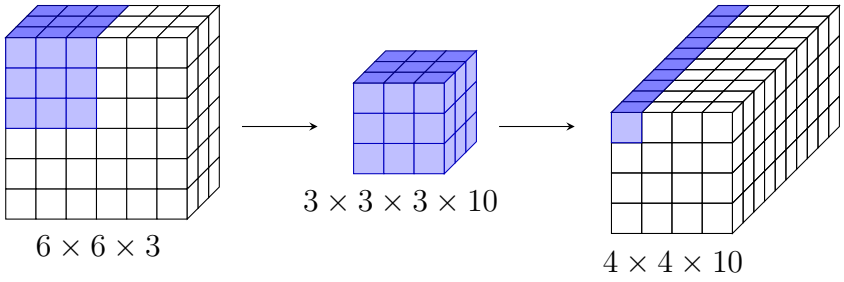
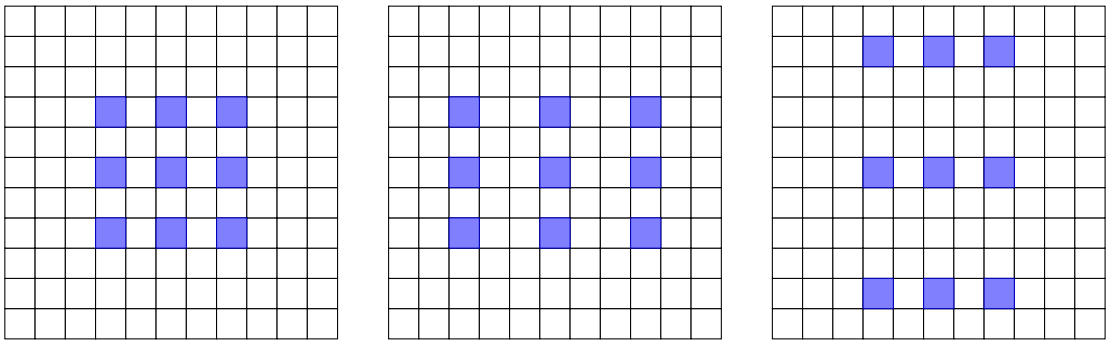


Figure 8. Illustration of 256 filter with kernel size 3×3 convolutional layer.

Atrous (dilated) convolutions Convolutional filters, as shown in Figure 8, consist of fixed kernel sizes such as 3×3 in the example. A convolution is said to be atrous(or dilated) if the kernel has spacing among the pixels. An example illustration is given in Figure 9. The spacing, so-called *rate* of the dilation, can be chosen distinctly for each dimension. Such convolutions expand the receptive fields without losing resolution or coverage [37]. For that reason, atrous convolutions are among key components for deep learning approaches to semantic image segmentation and used in many of the recent architectures such as [30, 2, 33].



(a) $rate = 2 \times 2$ (b) $rate = 2 \times 3$ (c) $rate = 4 \times 2$

Figure 9. Illustration of 3×3 atrous convolutions at different rates.

Depth-wise separable convolutions Made popular in [18], depth-wise separable convolutions (DWSConv) are a factorized version of the previously described standard convolutional layers. The process for a DWSConv is twofold. First, a single-depth convolution operation is separately applied to each channel of the previous feature map. Since the convolutional operations have a single depth, the depth of the output remains the same in this step. Second, N number of point-wise convolutions are used to combine the outputs of the depth-wise convolution where N is the desired channel depth of the resulting feature map. A traditional convolution operation filters and combines the inputs in one step. But, splitting this process into two parts, where one layer for filtering and one layer for combining, shown to drastically reduce the computation time and the model size [18]. Figure 10 shows an example of the process.

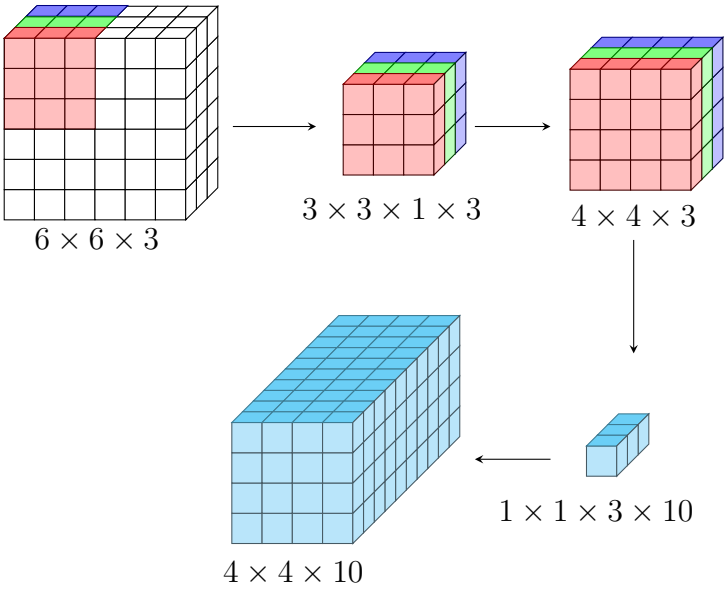


Figure 10. Depth-wise separable convolutional layer.

Batch normalization

Introduced in 2015, batch normalization [38] is a method in deep neural networks to boost the speed, stability, and performance. In neural networks, the distribution of inputs of each layer change during the training as the parameters of the previous layers are adjusted in each step. Thus, it is required to lower the learning rate during training and also carefully initialize the weights of each layer. Also, models with saturating non-linearities become harder to train. But, batch normalization allows us to tackle internal covariate shift and the strength of the method comes from applying normalization for each training mini-batch. Furthermore, batch normalization reduces the training time of state-of-art models (if not already used) by a significant margin. This results in being able to use higher learning rates, removal of drop-out layers, and affordability of application of additional modifications. [38]

After the demonstrated success of batch normalization method on the existing models in [38], it has quickly become a standard methodology for the successive models such as [30, 1, 2, 18, 21, 19].

Channel split Exploited in [21], channel split is a simple layer. Suppose an input with c number of channels, the operation splits this input into $c - c'$ and c' channels.

Channel Shuffle Proposed by [20], channel shuffle is an operation where a feature map of TN_c, where N_c is the number of channels, first formed into $T \times N_c$, second transposed to $N_c \times T$, and finally reshaped into N_cT. Figure 11 visualizes the operation. Furthermore, channel shuffle operation can be used in end-to-end training as it is differentiable.

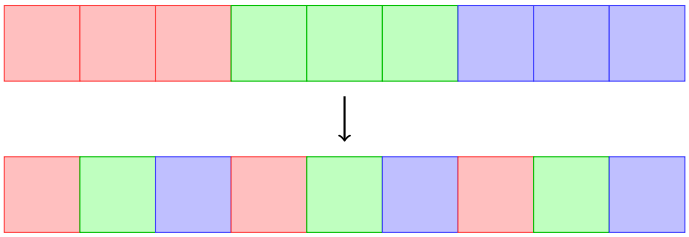


Figure 11. Visualization of channel shuffle layer.

Feature extraction and decoding

In computer vision and machine learning, especially in pattern recognition task, the patterns in the sensor data require description in a compact form. These representations, so called features, are a lower dimensional representation based on the pattern description. Features should ideally only contain the relevant information that relates to a pattern. The features have an important influence on the accuracy of the classification, time needed for classification and number of samples needed for training. Feature extraction is the task of reducing the dimensionality of pattern representation by finding mathematical methods. [39]

Over recent years, many deep neural network (DNN) based feature extraction methods have been proposed and used in the task of semantic image segmentation. As an example, Xception [40] network is a powerful feature extraction backbone that has been used in DeeplabV3 [30] with an atrous spatial pyramid pooling (ASPP) layer and in DeeplabV3+ [1] with both ASPP and U-Net [41] style skip-connections. It is also shown in DPC [2] that a more efficient feature extractor can benefit in search of architectures due to their shorter training time and correlation of accuracy with a deeper feature extractor such as Xception.

There are two main modules of a learning-based semantic image segmentation: an encoding module that performs feature extraction and a decoder module that recovers the spatial information. A typical example of such network is U-Net [41] which extracts lower level information in each step in the encoder path and then gradually recovers the spatial information during the decoding path using skip-connections that concatenate features that has the same dimensionality. An example of such architecture is shown in Figure 12. U-Net style skip-connections shown to obtain sharper boundaries of the objects. On the other hand, ASPP encoding method, as in DeepLabV3 [30], can capture contextually rich information by pooling features at different resolutions. An example of ASPP is shown in Figure 13.

Later, this extracted information can be reconstructed into the segmentation masks. In the case of U-Net [41], the information is upsampled using transpose convolutional layers where skip connections are added from the encoder features to the corresponding decoder activations. This method is a powerful technique to preserve the higher resolution information and shown to improve the performance of the network in [1]. Another approach is to simply apply bilinear upsampling which is a rather naive yet more efficient solution that was used in [30]. One can choose bilinear upsampling over complex upsampling designs to achieve a more efficient network by compromising on accuracy.

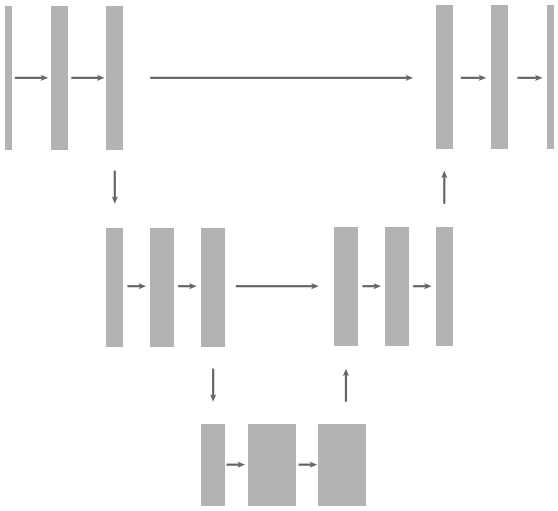


Figure 12. A visualization of U-Net style architecture with skip connections between downsampling and upsampling path.

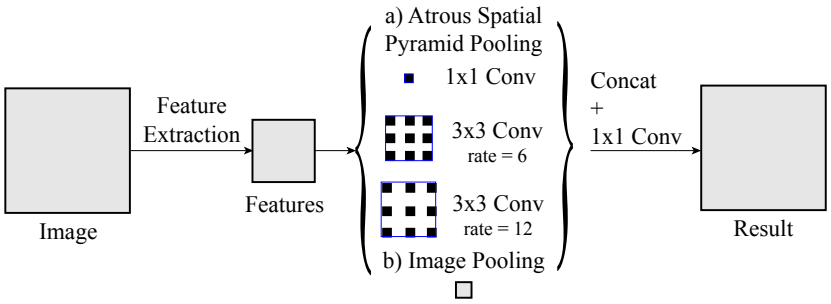


Figure 13. A simple example of atrous spatial pyramid pooling module.

2.2.3. Computational Efficiency

This subsection will discuss the computational efficiency of deep learning models. The layers that are described above such as depth-wise separable convolutional layers, batch normalization are some available tools for optimizing a model. But, in addition to these tools, we will describe some guidelines to further improve the computational efficiency of the models proposed by ShuffleNet V2 [21].

CNN models for semantic segmentation proposed in the field have been the top-performing approach in many benchmarks such as [15, 16, 17, 14]. But these approaches use deep feature generators and complex reconstruction methods for the task, thus making them unsuitable for mobile use, especially for the application of autonomous cars where resources are scarce and computation delays are undesired [33].

ShuffleNet V2 [21] as one of the recent proposals in feature generation demonstrates significant efficiency boost over the others while performing accurately. According to [21], there are four main guidelines to follow for achieving a highly efficient network design.

1. When the channel widths are not equal, there is an increase in the memory access cost (MAC) and thus, channel widths should be kept equal.

2. Excessive use of group convolutions should be avoided as they raise the MAC.

3. Fragmentation in the network should be avoided to keep the degree of parallelism high.

4. Element-wise operations such as ReLU, Add, AddBias are non-negligible and should be reduced.

To achieve such accuracy at low computation latency, they point out two main reasons. First, their guidelines for efficiency allow each building block to use more feature channels and have a bigger network capacity. Second, they achieve a kind of "feature reuse" by their approach of keeping half of the feature channels pass through the block to join the next block.

Another important issue is the metric of performance for convolutional neural networks. The efficiency of CNNs is commonly reported by the total number of floating point operations (FLOPs). It is pointed out in [21] that, despite their similar number of FLOPs, networks may have different inference speeds, emphasizing that this metric alone can be misleading and may lead to poor designs. They argue that discrepancy can be due to memory access cost (MAC), parallelism capability of the design and platform dependent optimizations on specific operations such as cuDNN's 3×3 Conv. Furthermore, they offer to use a direct metric (e.g., speed) instead of an indirect metric such as FLOPs.

2.3. Related Work

CNNs have shown to be the state-of-art method for the task of semantic segmentation over the recent years. Especially fully convolutional neural networks (FCNNs) have demonstrated great performance on feature generation task and end-to-end training and hence is widely used in semantic segmentation as encoders. Recent work in the field of semantic segmentation [31, 30, 1, 2, 32, 33, 34] follows the trend of using FCNNs. Moreover, memory friendly and computationally light designs such as [18, 19, 20, 21], have shown to perform well in speed-accuracy trade-off by taking advantage of approaches such as depth-wise separable convolution, bottleneck design, and batch normalization [38]. These efficient designs are promising for usage on mobile CPUs and GPUs.

Semantic segmentation as a real-time task has gained momentum on popularity recently. ENet [34] is an efficient and lightweight network offering low number of FLOPs in the design and ability to run real-time on NVIDIA TX1 by taking advantage of bottleneck module. Recently, ENet was further fine-tuned by [31], increasing the Cityscapes mean intersection over union (mIOU) from 58.29% to 63.06% by using a new loss function called Lovasz-Softmax [31]. Furthermore, SHUFFLESEG [32], demonstrates different decoders that can be used for ShuffleNet, prior work to ShuffleNet V2, comparing their efficiency mainly with ENet [34] and SegNet [42] by FLOPs and mIOU metrics but they do not mention any direct speed metric that is

suggested by ShuffleNet V2 [21]. The most comprehensive work on the search for an efficient real-time network was done in [33] and they report that SkipNet-ShuffleNet combination runs 15 frames per second (fps) with an image resolution of 640×360 on Jetson TX2. This work again, like SHUFFLESEG [32], is based on the prior design of the channel shuffle based approach.

DeepLabV3+ DPC [2] achieves state-of-art accuracy when it is combined with their modified version of Xception [40] backbone. In their work, MobileNet V2 has shown to have a correlation of accuracy with Xception [40] while having a shorter training time, and thus it is used in the random search [43] of a dense prediction cell (DPC). In this thesis work, the network developed for scene understanding task is inspired by the accuracy that they have achieved with the MobileNet V2 backbone on Cityscapes set in [2], and their approach of combining atrous separable convolutions with spatial pyramid pooling in [30]. To be more specific, we use lightweight prediction cell (denoted as basic) and DPC which were used on the MobileNet V2 features, and the atrous separable convolutions on the bottom layers of feature extractor in order to keep higher resolution features.

3. AUGMENTED REALITY

Augmented reality is a term that was brought upon in 1990 by Tom Caudell [44], a Boeing researcher, but the idea and the technology were already being studied since Ivan Sutherland's head-mounted three-dimensional display [45] in 1968. Recently, it caught the public attention by Google Glass, Microsoft Hololens, and Augmented Reality for iOS [46] but what exactly is defined as "Augmented Reality (AR)" and how is it different from "Virtual Reality (VR)"?

This chapter aims to clarify the concept of AR. For this reason, the chapter begins with the definition of the AR and the problem formulation, then follows a brief history to the technology, and lastly, the current limitations of AR are discussed.

3.1. Problem Formulation

The term *Augmented Reality* (AR) is often used in different definitions [47]. Since its first usage as a term by Tom Caudell in 1990 [44], the definition of the term was somewhat not clear in the literature, especially on its relationship to *Virtual Reality* (VR) and transparent head-mounted displays. In a broad definition, AR is associated with "augmenting natural feedback to the user with simulated cues". [47] points out two questions with regards to the specific definition of AR.

1. What is the relationship between AR and VR?
2. Is AR, as a term, only limited to see-through head-mounted displays?

The first question brings out an important clarification. AR and VR are in fact related and it is logical to think the couple along with each other. However, there is a definite difference between the two. The user of a VR system is completely immersed in a fully fabricated world, in which this world might or might not mimic the real-world properties and may invalidate the physical bounds of these properties such as gravity, time and material attributes. On the other hand, a real-world environment must follow the laws of physics. According to [47], these two definitions form a *continuum* what is referred to as *Reality-Virtuality continuum*. Figure 14 shows a representation of the continuum.

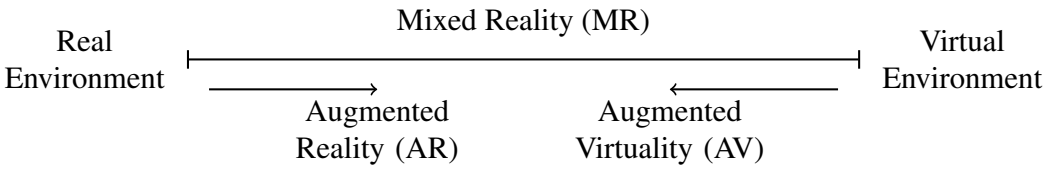


Figure 14. Reality-Virtuality continuum.

Figure 14 addresses the first question, however, the second question needs further explanation. The broad definition of the AR mentioned above, which was "augmenting natural feedback to the user with simulated cues", already invalidates the claim of limiting the usage of the term AR to only head-mounted see-through systems.

Furthermore, [47] presents two cases of AR systems which clarify the misconception of the second question.

1. “See-through” AR displays where the display has the means to show the real-world as see-through, thus achieve the utmost degree of “realspace imaging”, and display computer-generated objects on top of the display.

2. Monitor-based AR displays are non-immersive, in other words, window-on-the-world, form of AR where computer generated images are superimposed over live or stored video images.

The clarification on the topic is important as the work of this thesis falls to the second category mention above. Although more explanation of the system will be given in the following chapter, AR on mobile phones are considered to be monitor based AR displays.

To sum up, AR is a term in which the user is able to experience the real-world while virtual cues are superimposed upon or composited with the real world. Because of this, AR complements the reality, unlike VR where the reality is replaced with virtuality [5]. Moreover, AR is located in the middle of the Reality-Virtuality continuum. An example of an AR environment is shown in Figure 15.

Figure 15. An example image of an augmented reality environment where virtual and real objects co-exist[1].

[1]Image is taken from “10.000 Moving Cities – Same but Different, AR (Augmented Reality)” by Marc Lee, licensed under `https://creativecommons.org/licenses/by-sa/4.0/deed.en`

3.2. A Brief History of Augmented Reality

This section aims to give a brief historical overview of the AR and its development. Since the history of AR dates back to 1968, only the major breakthroughs in the field are mentioned in this section.

The earliest AR system was developed by Ivan Sutherland. The system in his work was a head-mounted see-through display which overlaid simple wire-frame drawings. The so-called head-mounted three-dimensional display was impractical for public use due to the technology at hand. Later, Myron Kruger created VIDEOPLACE [48], an artificial reality laboratory, that used video cameras and projectors to create silhouettes of users in an interactive environment in 1974. However, the term "Augmented Reality" was first used by Tom Caudell, a Boeing researcher, in his work [44] in 1990.

Although there was progress in the field for more than 20 years, the first properly functioning AR system can be considered the Virtual Fixtures [4] that was developed at USAF Armstrong's Research Lab by Louis Rosenberg in 1992. Since the 3D graphics were not fast enough to achieve photorealism in the 1990s, Virtual Fixtures consisted of two physical robots that were controlled by the user via a worn exoskeleton controller. By the aid of a pair of binocular magnifiers, these robot arms were brought to the position of the user's hands to fully immerse the user in the experience. The system also had virtual graphics that were superimposed to the vision of the user as shown in Figure 16. The system proved to increase human performance significantly. [4]

Figure 16. Virtual fixtures used to increase user performance in the task of telerobotic control of Fitt's Law peg-board[2].

After Virtual Fixtures, Julie Martin created the first AR theatre production called "Dancing in Cyberspace" in 1994 that consisted of acrobats who danced with the virtual objects on their physical stage. After that, with the advancing technologies by the end of the 1990s and early 2000s developments such as "Battlefield Augmented

[2]Image by AR Trends licensed under https://creativecommons.org/licenses/by-sa/4.0/deed.en taken from https://commons.wikimedia.org/wiki/File:Virtual-Fixtures-USAF-AR.jpg

Reality System (BARS)" [49] by The Naval Research Lab, and open-source framework ARToolKit for overlaying computer graphics to a video camera feed accelerated the development of the AR. In 2012 with the release of Google Glass [50], Microsoft HoloLens [51] development edition in 2016 showed that the augmented reality as a portable technology is possible.

After these advancements in the field, AR has moved to the mobile phones with the release ARKit API tool by Apple in 2017. Although the technology was already on the mobile phones, the development of AR applications was hard due to the lack of an official framework. Following the ARKit, Google released the ARCore [3] within the same year, making AR development accessible for the Android developers. These frameworks took advantage of the many sensors that come with modern smartphones such as accelerometer, gyroscope, camera. Using such sensors the ARKit and ARCore were able to perform online simultaneous localization and mapping (SLAM) on the mobile phones. SLAM will be discussed in detail in the following sections.

The following section will introduce some concepts that are used in AR, especially on mobile devices to locate the device and understand the surroundings.

3.3. Matching the Real-World with the Virtual-World

Augmented reality, especially if the environment is not modeled beforehand, is a difficult task. The camera position should be able to be located in a mapping of the current environment and the surfaces and points should be mapped in three-dimensional space to properly add objects to the scene by creating a sense of reality from computer-generated images over the actual reality.

Such a system, in its current state of technology, should be able to find out the real world coordinates from the sensory information that are present such as camera images. To achieve this, referred to as image registration, numerous computer vision methods which are mostly on video tracking are applied [52]. These computer vision methodologies are often rooted from visual odometry.

The first step in such systems is to detect interest points, fiducial markers or optical flow within the view of the camera which is usually performed by feature detection algorithms [53]. Such algorithms are corner detection, thresholding (e.g. to detect bright red led markers), blob detection and so on. In the second step, we restore the real world coordinates from the information that is obtained from the first step by applying tracking various algorithms [54, 52].

The next subsection will examine the mentioned tracking methodologies that are used in the second step in more detail.

3.3.1. Tracking

Tacking is the task of progressive determination of dimensional properties at runtime. In an AR system, position and orientation properties of entities such as user's head, limbs or the AR device (e.g. the camera) can be tracked. [54]

To define it in detail, *tracking* is the term to express dynamic sensing and measuring of AR systems. Unlike the tracking in 2D space, which is a common tradition

in computer vision, AR systems require 3D tracking, thus the term *tracking* is synonymous to the 3D tracking in the context of AR. In such a system, where objects are displayed in 3D real-world, continuous calculations for pose estimations should be updated by tracking where these estimations include position and orientation of the real entities. [54]

There are two important concepts on the topic of tracking in the context of AR. First, the devices that will do the measurements should be compared and adjusted to give the same results. This process is referred to as *calibration* and its objective is to determine the parameters of a device from a known reference device so that the measurements delivered by the device to be calibrated are on the known scale. This is an important part of AR systems and a prerequisite of *tracking* and *registration*. Calibration may be done once during or after the manufacturing of the device, every time before starting the operation or concurrently during the operation which is often referred to as auto-calibration. [54]

Second, an AR system should align the coordinate systems of the real world and the virtual world. This is referred to as *registration* in the context of AR. In order to do so, the user's head or the camera that is providing the video feed should be tracked. There are two types of registration. When the camera position is stationary, then calibration of the tracking system is required which is referred as to static registration. On the other hand, when the camera is moving, then it is referred as to dynamic registration and this process requires tracking. [54]

AR systems usually depend on computer graphics pipelines such as OpenGL to display overlays on the real world. Such kind of pipeline where there is a model transformation, view transformation, and projective transformation is a common process independently from the type of the AR display [54]. In the case of mobile phones, the frames are captured, rendered and composited using these transformations. As previously mentioned, registration helps us to achieve a matching transformation between the real-world and the virtual object.

Model transformation The model transformation defines where an object located in the environment. It is the transformation information of this local object relative to the 3D global word coordinates. So in an AR system, if one wants to place a virtual object in the environment, then its model transformation should be tracked. This transformation contains the information of 3D coordinates and 3D orientation. [54]

View transformation In an AR system, tracking the view transformation is of utmost importance. This transformation defines the relationship of 3D camera coordinates and the 3D global world coordinates. In the case of mobile phones, where the category of the display falls into the see-through video displays, one transformation for the position of the camera may be sufficient, unlike the stereoscopic displays where we may need calibration of the camera and the display separately. [54]

Projective transformation Since the display is in 2-dimensions, we need another transformation for transforming from the 3D transformation that we have acquired so far. Projective transformation is the relationship of 3D camera coordinates and the 2D

device coordinates. This is usually done by mapping to a unit cube to the screen and then dropping the z-axis after the view transformation. This transformation is normally calibrated offline and should be done for each camera and each display separately.

In order to obtain the view matrix, one should be able to localize the camera position in the environment. This thesis is concerned with the mobile phone subset of the AR where the display medium is a video see-through display. There are several methods to localize the device position within the real world. In the case of no prior model of the environment, a methodology called *visual odometry* can be used to locate the view position. In brief, visual odometry, simultaneous localization and mapping (SLAM) [55], is the continuous tracking of camera's position and orientation (position and orientation together are referred as pose) relative to an arbitrary starting point. It calculates a 3D reconstruction of the environment. [54]

Visual odometry flow usually contains the following steps:

1. Finding interest points in the initial frame by using an algorithm such as Harris [56], LBP [57] and so on.

2. Track the interest points that were found in the previous frame in 2D using for example KLT [58].

3. Compute the essential matrix between the current and previous frames from the matching features with a five-point algorithm such as Nistér's [59] within a RANSAC [60] loop.

4. Form the incremental camera pose from the essential matrix.

5. The essential matrix establishes the translation of the pose up to scale and this scale should be estimated separately to keep it coherent throughout the sequence of tracked images. To compute the scale and achieve the coherence, one can use an approach called structure from motion (SFM) where 3D point observations of the same feature are triangulated over time.

6. Evaluate the next frame.

In addition to visual odometry, smartphones, thanks to their additional sensor, can take advantage of the inertial information provided by the gyroscope and accelerometer sensors. Fusing this additional information to the information that is obtained by the visual odometry, visual-inertial odometry (VIO) [61], can allow processing the keyframes at lower rates thus allowing performance gains on the mobile devices where the resources are scarce and the efficiency is at utmost importance.

Today, ARCore [3] and ARKit [46] uses visual-inertial odometry at their core to achieve 3D scene reconstruction and provide anchors that can be tracked in real 3D world space. This thesis work is exploiting the functionality that is offered by Google's ARCore to further improve upon the scene understanding that will be provided by the semantic image segmentation which we have discussed in Chapter 2. Specifically, we use the 3D tracking functionality that is offered by ARCore is used in the tracking of the segmentation masks to avoid loss of positioning between the frames that pass during the time frame of the segmentation mask calculation.

3.4. Limitations

This section will give an outlook of the limitations in the augmented reality systems. Before AR becomes a part of a regular person's everyday technology, some of the challenges in the field such as costs, weight, power usage, ergonomics should be addressed.

3.4.1. Portability and Outdoor Use

According to [62], especially the over-head display type of augmented reality devices are often hard to carry for outdoor use. Furthermore, they claim that the technology is coming to the smart-phones in their work published in 2007.

Today, we see that the over-head display systems such as Microsoft HoloLens [51] are improvements to this challenge but portability and outdoor use is a still open challenge due to the cables and weather resistance.

On the other hand, we see that the technology on the smart-phones, with the power of ARCore [3] and ARKit [46] getting a stronger position in the field of AR. But the challenges such as low-light environments, field of view limitations, mobile CPU and GPU limitations still an open issue. The hybrid tracking algorithms such as visual-inertial odometry are heavy tasks to be run on the current mobile CPUs and GPUs. Furthermore, due to the dependence on the gyroscope sensory data, using such systems in moving environments such as elevators may lead to drifting in the estimations. This kind of drifting is also possible in case of visual-inertial odometry method if there are a lot of moving objects in the scene.

3.4.2. Calibration

The mobile phones require calibration in order to be used in AR systems and calibration of the devices is still complicated and extensive. This issue may be solved by calibration-free or auto-calibrating approaches that minimize the initial set-up requirements. We see the persistence of this issue when we look at the limited number of supported devices of ARCore [3]. This is due to the lack of calibration of the old devices or due to the device manufacturer's will.

3.4.3. Latency

Systems delays are the primary source of dynamic registration errors according to [49] as cited by [62]. The latency can be reduced by using methods such as pre-calculation, temporal stream matching (video-see-through displays), and future viewpoint prediction. Furthermore, system latency of the devices can be lowered by careful design of the system. [62]

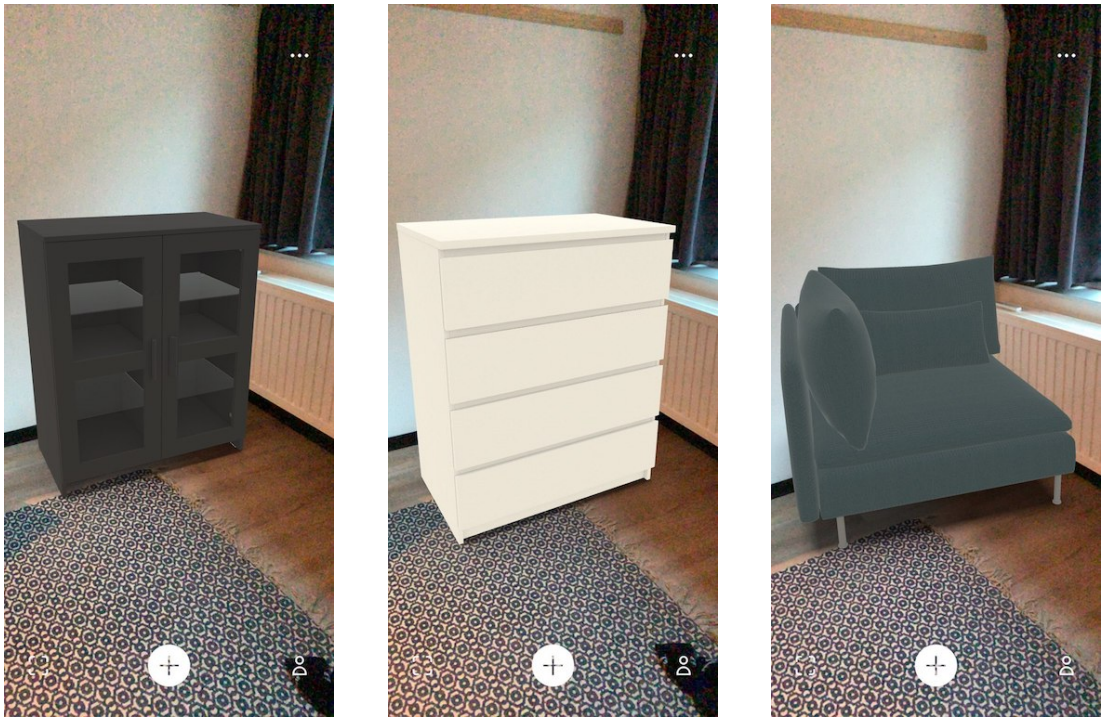


Figure 17. Adding virtual furniture to a home using IKEA Place [12] application. Light in the scene is estimated and applied to the virtual objects.

3.4.4. Overload and Over-reliance

The user interface of the AR systems is also an important aspect of the system. It should be designed such that it follows certain criteria to not overload the user. Furthermore, it should not let the user to overly rely on the information given by the AR system since it may cause the user to miss some important cues from the real world. The system, in general, should aim to improve the performance of the user while not causing loss of attention and becoming a source of distraction. [62]

3.5. Summary

This chapter has given a brief description and the related work on the technology of augmented reality. The problem was formulated and a brief history was given. Later, the chapter described the methodology to match the virtual world to the real world. Lastly, some limitations of the technology were presented.

Over the course of 50 years, since 1968, augmented reality is now possible on our daily devices such as Android or iOS phones. Today, although the field still has open challenges, it is easier for augmented reality to be used by everyone on their daily tasks today. For example, applications such as Houzz [11], and IKEA Place [12] allow you to design your house using augmented reality on mobile phones. Example screenshots from the IKEA Place application are shown in Figure 17.

In the context of this thesis work, augmented reality tracking method is exploited by using the fast 3D tracking of the points over the surfaces. The semantic segmentation

masks are then mapped to along the tracked points. Thanks to the generated semantic segmentation masks, which we have described in Chapter 2, online destructive segmentation of the scene is possible. These masks, combined with the power of tracking in a 3D environment, can achieve realistic and low latency destructive reality augmentation.

The next chapter will describe the implementation details on semantic segmentation and augmented reality.

4. IMPLEMENTATION

This thesis focusses on inpainting objects in augmented reality using semantic image segmentation for online scene understanding. To this extent, our first objective is to achieve the online scene understanding in a real-time manner using a deep learning network. Second, the thesis work aims to use this scene understanding for applying destructive scene augmentation (i.e. removing objects from the real world). The prime focus of the implementation is Android-based mobile devices with ARCore [3] support.

The device target, Android-based mobile devices, require an efficient solution to each step. The processing power on the smartphones are scarce, thus the thesis work aims to have an efficient solution during each step of the implementation. The first part of the chapter, namely deep learning based approach to semantic image segmentation, proposes a novel state-of-art efficient solution to the task. The solution will be discussed in detail in addition to the published work [63] in this thesis. Later, an augmented reality implementation is exploited to track the segmentation masks in 3D space for achieving smooth and stable overlay over the objects in the real world. The last part is a simple inpainting approach to demonstrate the applicability of the methodology that is being proposed in this thesis. The overall process is shown in Figure 18.

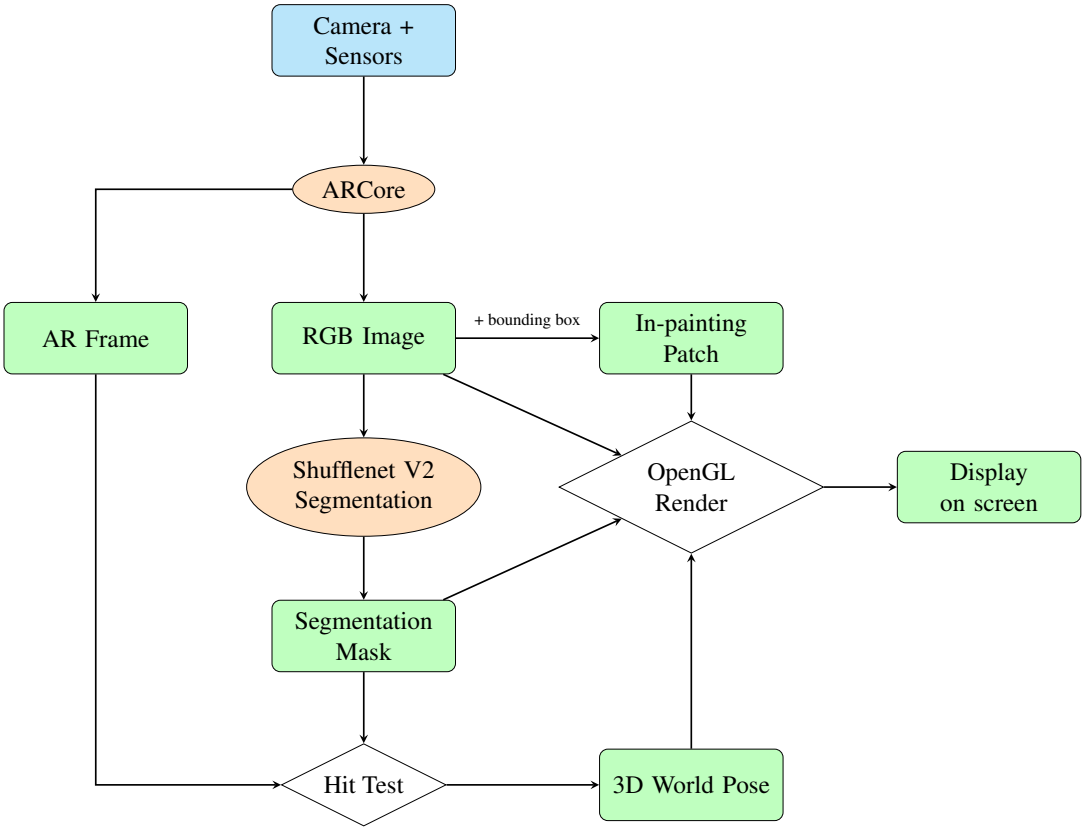


Figure 18. The pipeline of the destructive scene augmentation proposed in this thesis work.

This chapter consists of three main sections. Section 4.1 describes the network design, training, optimization and smart-phone implementation in detail. Next, the details of the augmented reality implementation on an Android-based smartphone is given in Section 4.2. Lastly, the implementation details of a simple inpainting method are described in Section 4.3.

4.1. Semantic Image Segmentation

Semantic segmentation is a challenging task of computer vision. Recent studies show that fully convolutional neural networks are the state-of-art solution to the task. Thus, we propose an efficient solution to achieve real-time semantic segmentation on mobile phones using an FCNN.

In short, the proposed FCNN consists of three fundamental architectures. First, to extract the features from the RGB images we use a powerful and efficient network, namely ShuffleNet V2 [21]. Then, the feature maps are fed into the Deeplab V3+ basic heads [1] or DPC [2] heads for increased accuracy. Finally, the resulting features decoded by a naive approach (i.e. bilinear upsampling) to produce the segmentation masks. Further details on the design of the network are given in Section 4.1.1.

The architecture is trained by supervised learning for selected classes. Over recent years, many datasets containing pixel-level annotations emerged in the field, thus opening new windows for scene understanding. Neural networks are performing the best when they are trained over numerous amount of data. For that reason, we train the network over several datasets such as COCO and Cityscapes to achieve the best accuracy which was shown in [1]. To demonstrate in the thesis work, three classes (keyboard, book, apple) has been selected for training. Details of the training schema are described in Section 5.1.2.

4.1.1. Network design

This section will describe the details of the proposed network. The approach is based upon the DeeplabV3+ [30] and DPC [2] architecture. However, we replace the feature extractor with an efficient feature extractor, namely ShuffleNet V2. As shown in Figure 19, the segmentation starts with the ShuffleNet V2 feature extraction. The final down-sampling factor of the feature extractor is denoted by $output\_stride$. Then the extracted features go through the encoder head which is either the simplified head which was proposed in [30] or the DPC encoder head which was proposed in [2]. Finally, resulting mask is adjusted to the target dimensions by a simple decoder which is bilinear upsampling and the segmentation result is obtained by applying an ArgMax operation.

For the task of feature extraction, ShuffleNet V2 was chosen due to its efficiency while achieving high accuracy. Our selection for the depth multiplier of ShuffleNet V2 is $\times 1$. This can be seen in the output channels column of Table 1. The decision was purely made by accuracy and speed results of this variation on the ImageNet dataset and experiments on different variations of the depth multiplier were not tested in this thesis work. Although, one might choose lower values for this hyper-parameter to

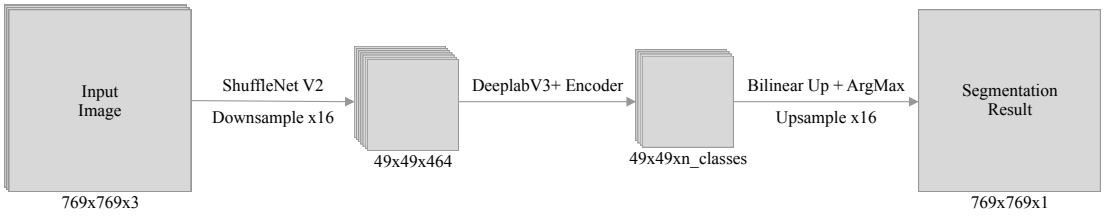


Figure 19. General view of the network at 769×769 with $output\_stride = 16$. $output\_stride$ denotes the down-sampling factor of the feature extractor.

achieve faster inference time by compromising on the accuracy and vice versa higher values might result in a gain of accuracy in favor of inference speed.

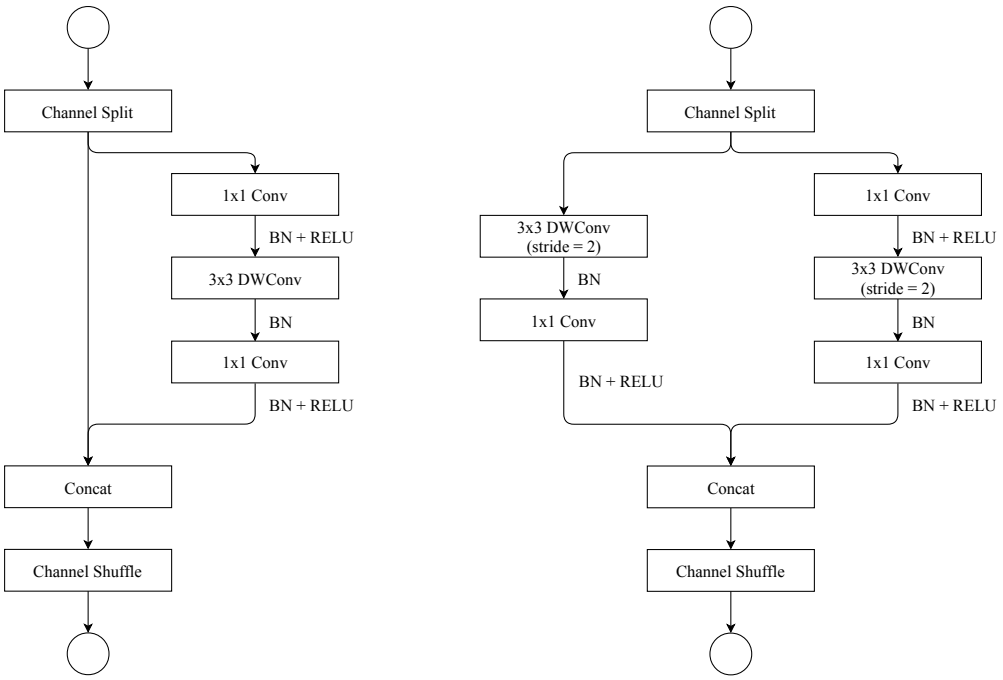
Table 1. Proposed network architecture at $output\_stride = 16$. **DLV3** denotes DeepLabV3+.

	Layer	Output Size	Kernel	Stride	Rate	Repeat	Output Channels
	Image	769×769					3
ShuffleNet V2	Conv2D	385×385	3×3	2		1	24
	MaxPool	193×193	3×3	2			
	Stage 2	97×97		2	1	1	116
		97×97		1	1	3	
	Stage 3	49×49		2	1	1	232
		49×49		1	1	7	
	Stage 4	49×49		1	1	1	464
		49×49		1	2	3	
DLV3	Encoder Head[1]	49×49		1	1	1	512
	Conv2D	49×49	1×1	1			256
	Conv2D	49×49	1×1	1			$n\_classes$
	Bilinear Up	769×769					$n\_classes$
	ArgMax	769×769					1

[1] DPC or DeeplabV3 basic head.

Figure 20 shows the stages of the feature extractor architecture in detail. Each stage consists of one spatial down-sampling unit and several basic units. In the original implementation of ShuffleNet V2 [21], $output\_stride$ goes as low as 32. In our approach, the entry flow, Stage2, and Stage3 of the proposed feature extractor are implemented as in [21]. In the case of $output\_stride = 16$, the last stage, namely Stage4, has been modified by setting $stride = 1$ instead of 2 on the down-sampling layer and the atrous rate of the preceding depth-wise convolutions are set to network stride divided by $output\_stride$ as described in [1] to adjust the final down-sampling factor of the feature extractor. However, in the case of $output\_stride = 8$, stride and atrous rate modification starts from Stage3. We choose to focus on $output\_stride = 16$ due to its faster computation speed.

After the features are extracted, we employ the DeeplabV3+ encoder head (or DPC head). The architectures of both encoder heads are shown in Figure 21. The basic variant does not contain atrous convolutions in its design since it is designed lower complexity, thus result in faster inference speed. On the other hand, five different



(a) Basic unit (b) Down-sampling unit

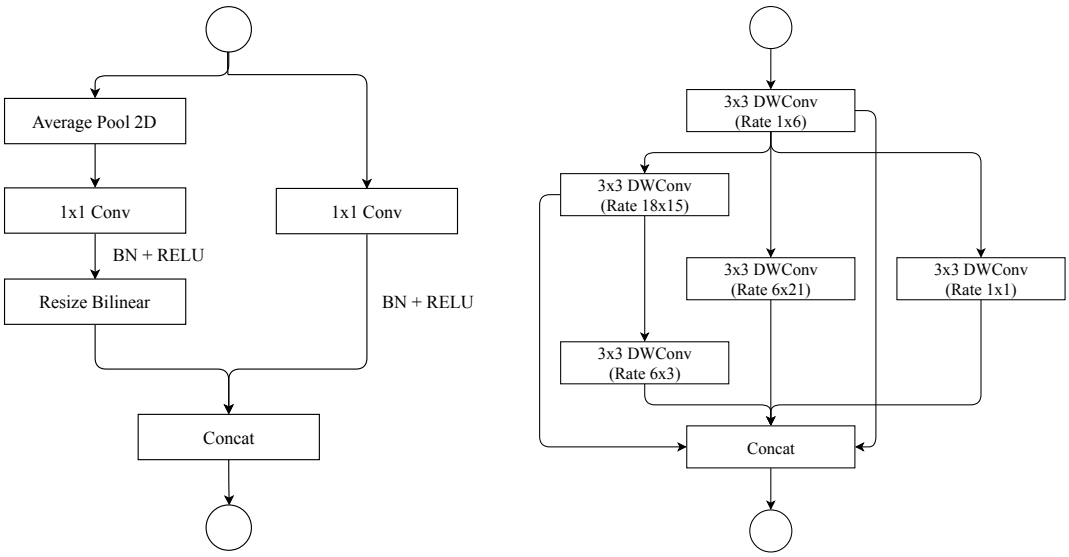
Figure 20. ShuffleNet V2 units. (**DWConv** means depthwise convolution.)

depth-wise separable convolutions at different dilation rates are used in the architecture of DPC encoder head. As a part of the exit flow, as seen in Figure 22, encoded features then goes through a 3×3 convolution of $depth = 256$ which are then reduced to $depth = n\_classes$ with the final convolutional layer. A dropout layer with a keeping-probability of 0.9 is applied afterward.

For the decoding part, the architecture uses a naive approach as [30], where we use a simple bilinear upsampling layer to adjust the probabilities to the target dimensions. In the case of $output\_stride = 16$, the upsampling factor is 16. It should be noted that the resizing should be done before the ArgMax operation as suggested in [1]. In [1], a decoder unit with a skip connection from an earlier stage of the network (e.g. when the $output\_stride = 4$) is proposed to be concatenated to the encoded features to preserve finer details in the segmentation result. We have not included this part as it adds extra inference time to the network and increases the complexity.

4.1.2. Smartphone implementation

For the implementation of the neural network, this thesis work uses Tensorflow Library [64]. In order to use trained models on mobile devices such as Android or iOS-based smartphones, Tensorflow offers Tensorflow Lite [65]. Tensorflow Lite can deploy machine learning models on mobile and IoT devices. It is an open-source deep learning framework for on-device inference. In a sense, it is a smaller subset of Tensorflow, meaning that the essential functionality for inference is provided but complex operations might be missing in implementation.



(a) Basic encoder head. (b) DPC encoder head.

Figure 21. Encoder head variants.

Figure 22. Exit flow.

In this thesis work, we export the trained models as a Tensorflow Lite compatible model. The operations and layers that are used in the proposed network are already supported by the framework thus require no custom operations. The model is then added to an Android application project that uses Tensorflow Lite Android framework in order to perform on-device inference.

4.2. Augmented Reality

Augmented reality is a major area of interest within the field of computer vision. Many applications of AR has shown to increase the performance of the user and also, it is widely used for entertainment purposes as well. One of the recent frameworks that offer ease of development of AR applications by Google, ARCore [3], is used in this thesis work for 3D localization and tracking of the segmentation masks. Furthermore, later in the chapter we will show how the combination of semantic segmentation and ARCore can be exploited to perform destructive scene augmentation.

The ARCore framework is used in two aspects in the context of this thesis work. First, the 3D hit test where the 2D projection coordinates are converted to the 3D points in the world model. Second, these localized points are tracked in 3D world space using the ARCore. Later, these points, camera and projection matrices and the segmentation mask is sent to OpenGL for rendering the points in the 3D model of the real world over the video display.

The motivation to implement 3D tracking using ARCore in our system is three folds.

- There is a computation delay in the process of generating the segmentation map, thus the frames that past during the inference time is not segmented.
- Running deep learning models on devices such as smartphones are battery consuming and less battery consuming tasks such as tracking can be exploited to lower the rate of segmentation mask generation.
- Our primary focus devices, smartphones, are usually handheld, thus the camera view is not stationary. For that reason, the user might be moving the device to look at another place. Because of the segmentation mask generation delay, the mask should be properly moved according to the 3D movement of the device.

4.2.1. Mask Localization and Tracking

As the first step in the AR module of the implementation, we locate the generated segmentation mask in the 3D model of the real world environment. This is an important step as it is crucial to achieving steady masking during the augmentation.

The ARCore provides the functionality to perform hit tests easily. The framework automatically detects the flat surfaces, which is referred to as a *plane*, within the environment. These surfaces are one of the *trackable* entities of the framework. It is possible to perform a hit test to a trackable within the scene. The hit test performs a ray cast from the device in the direction of the given point in the camera view. Meaning that the 2D locations on the device screen can be converted to 3D points in the environment.

In the implementation, we first determine the bounding box of the biggest connected component in the segmentation mask. To find the biggest connected component in the segmentation mask the thesis implementation uses functions provided by the OpenCV [66] library. As a preprocessing step, we perform a *morphological open* followed by *morphological close* operation with a rectangular structuring element of size 5×5. Afterward, the biggest component within the mask is found and the bounding box of this component is extracted.

By exploiting the capability of ARCore, we can specify coordinates on the screen of the device and get the corresponding plane intersections, thus it is possible to obtain 3D points of the mask in the real world for rather flat objects. The four points found in the previous bounding box determination are used for hit test on the frame. If the all for points of the bounding box cause a hit on the same trackable plane, then these points are turned into trackables and the location of the segmentation mask in the system is updated. As long as the ARCore is in tracking state references to the trackables can be used for acquiring pose updates. These world coordinates are then used for rendering the segmentation mask on the screen. The flow chart of the process is shown in Figure 23.

In addition to the hit test, ARCore provides the transformation matrices for the view and the projection. This information will be needed during the rendering process. On each frame, the view and projection matrices are obtained from the framework and sent to the renderer implementation. The rendering details will be discussed in the following subsection.

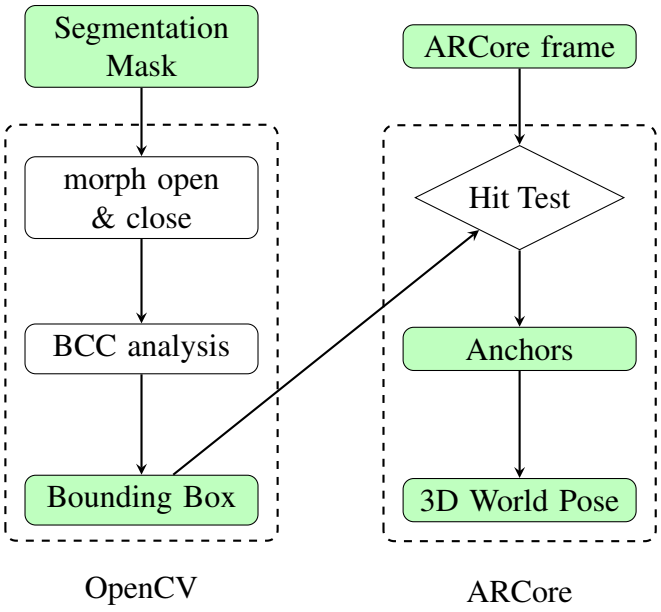


Figure 23. Overview of the tracking method. **BCC** denotes biggest connected component.

4.2.2. OpenGL rendering

After the world pose information of the segmentation mask bounds are obtained, OpenGL is used for rendering the graphics on the screen. Android smartphones use a subset of OpenGL computer graphics rendering application programming interface called OpenGL for Embedded Systems (GLES). The flow of the GLES pipeline is shown in Figure 24. This subsection will describe the implementation details of the buffer object and texture preparations, vertex shader, and the fragment shader. On each frame, the whole pipeline is executed to produce the image on the screen of the device.

The first step in the rendering process is sending the vertices and the texture positions along with the textures as it can be seen in Figure 24. To do so, the points that were passed to the renderer first combined into a 1-dimensional array of (x, y, z) values, then uploaded to the vertex shader along with the corresponding texture coordinates of the segmentation mask. These values fall into the range of $[-1, 1]$ in the GLES coordinates where the point $(0, 0)$ being the middle of the screen. However, the texture coordinates are ranged from $[0, 1]$ where point $(0, 0)$ being the bottom left corner.

After the coordinate upload to the vertex shader, the textures are sent to the fragment shader. We send two different textures to the shader. The segmentation mask which consists of one channel and the color image of the original scene to perform the inpainting in the fragment shader. These textures are sent after the segmentation mask is generated and the hit test is performed by using GLES interface function *glTexImage2D*.

The last step of the preparation is to calculate and set the model view projection matrix. The view and the projection matrix is provided by ARCore. As the model matrix, we are using an identity matrix. The following equation shows the transformation of the 4×4 matrices into model view projection matrix which is denoted as MVP:

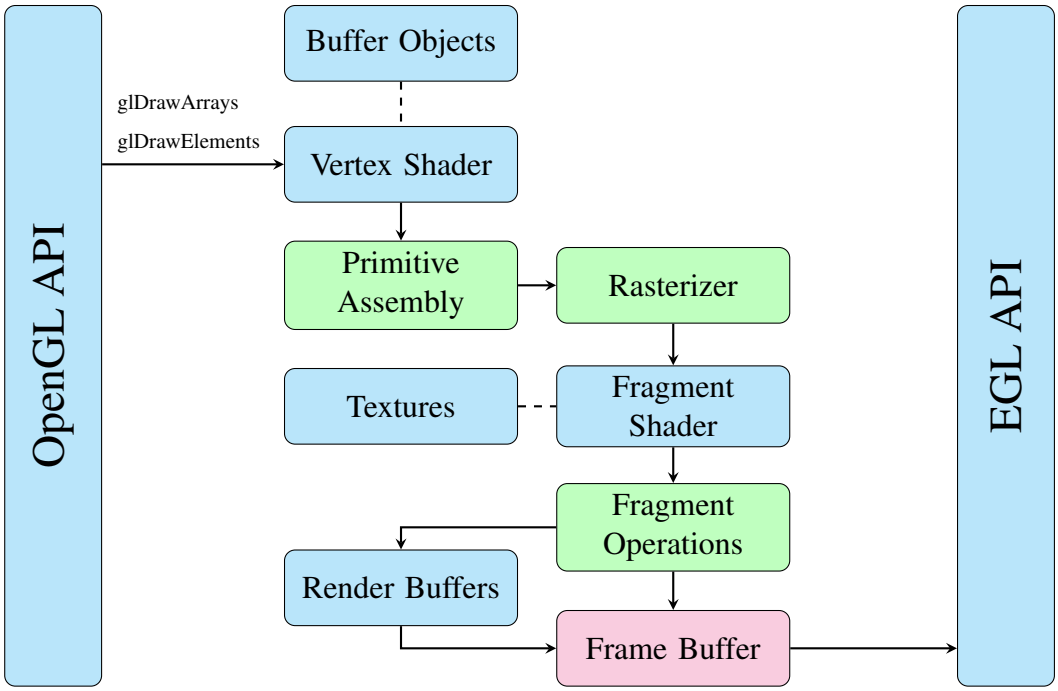


Figure 24. OpenGL ES 2.x pipeline. **Blue node:** You have to set, **Green node:** You do not have any control, **Magenta node:** Optional.

$$\text{MVP}_{4\times4} = \text{projection}_{4\times4} \times \text{view}_{4\times4} \times \text{model}_{4\times4} \tag{3}$$

After the preparation steps, the GLES function *glDrawElements* is used to render the inpainting or the segmentation mask on the screen. The elements in GLES are drawn as triangles. Since our segmentation map is a plane, we draw 2 triangles on the screen. The triangle edges are called indices to determine the order in which the points are drawn. For example, before calling *glDrawElements*, one can set the indices as $[0, 1, 2]$ so that the vertices that were set before (the coordinates of the edges) form a triangle using three edges where the first edge is the first vertex and the second edge is the second vertex and so on. These vertices are processed by the vertex shader of our implementation which can be seen in Listing 4.1. The MVP transformation is applied to each of the vertices and the texture coordinates are sent to the fragment shader.

Listing 4.1. Vertex shader implementation in GLES version 3.

```
#version 320 es

in vec3 aPosition;
in vec2 aTexCoord;
uniform mat4 uModelViewProjection;

out vec2 vTexCoord;

void main() {
    gl_Position = uModelViewProjection * vec4(aPosition, 1.0);
    vTexCoord = aTexCoord;
}
```

The fragment shader handles individual pixel colors that will be drawn on the screen. As seen in Listing 4.2, implementation consists of two different modes for rendering and two textures, where the *sTexSegmentation* is the segmentation mask and *sTexImage* is the real world image. The rendering mode is set by the user in the application. The first mode, denoted as SEGMENT, renders the segmentation mask as a red overlay over the screen. On the other hand, the second rendering mode, denoted as PATCH, renders the real world inpainting patch over the segmentation area determined by the segmentation mask. The resulting color of the pixel is denoted as *fragColor*.

Listing 4.2. Fragment shader implementation in GLES version 3.

```
#version 320 es
precision mediump float;

#define SEGMENT 1
#define PATCH 2

in vec2 vTexCoord;
out vec4 fragColor;

uniform sampler2D sTexSegmentation;
uniform sampler2D sTexImage;
uniform int renderMode;

void main() {
    vec4 color;
    switch (renderMode) {
        case PATCH:
            vec4 segColor = texture(sTexSegmentation, vTexCoord);
            color = texture(sTexImage, vTexCoord);
            fragColor = vec4(
                color.rgb,
                segColor.r == 0.0 ? 0.0 : 1.0
            );
            break;
        case SEGMENT:
            color = texture(sTexSegmentation, vTexCoord);
            fragColor = vec4(color.rgb, color.r);
            break;
    }
}
```

4.3. Inpainting

The inpainting is performed within the fragment shader so there is no additional overhead of calculating the inpainted version of the scene. The segmentation mask sent to the shader marks the coordinates to be inpainted as a value of 1 and the locations that are not to be inpainted is set to 0. As seen in the implementation code Listing 4.2, when the rendering mode is set as PATCH, the algorithm checks if the corresponding location lays within the area to be inpainted. If that is true, the alpha channel of the *fragColor*, which is the fourth channel, is set to 1 meaning that the texture in this pixel

should be drawn as opaque. Otherwise, if that is false, the pixel should be rendered as fully transparent by setting the alpha channel to 0.

The *sTexImage* contains a patch of the real world image from a rectangle around the bounding box of the segmentation mask. In order to achieve a natural looking inpainting, we search for 20×20 sized patches around the bounding box of the segmentation mask and choose the one that has the lowest standard deviation among the color values. This method is a naive approach to inpaint an image. Our aim is to find a patch that has a rather flat surface near the object to be segmented and fill the mask with the found patch.

Figure 25 shows the flow of the inpainting implementation. Once we obtain the bounding box from the segmentation mask, we perform hit test on the ARCore frame. If the hit test fails, meaning that any of the four corners of the box did not intersect with a plane or intersections were in different planes, then we skip the segmentation mask and the frame. If the hit test was successful then we run the patch finding algorithm and extract the texture from the image. The texture is then uploaded to the GPU on the call of *onDraw* function of OpenGL and drawn on the screen by the shader implementations which are shown in Listing 4.1 and Listing 4.2.

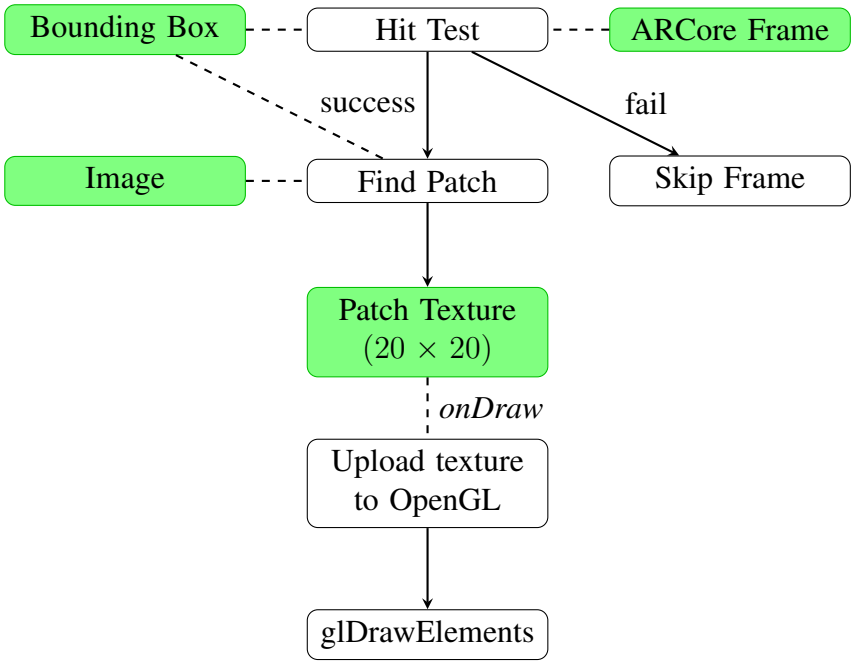


Figure 25. Flowchart of the inpainting method.

4.4. Limitations

This section describes some of the limitations of the implementation.

1. The environment is mapped on-flight by ARCore, thus the system requires user cooperation for initial detection of the planes in the environment.

2. Since the points are located on the plane, the current implementation assumes that the object is fairly planar, meaning that the object has a low height.

3. The segmentation calculation has a calculation overhead, thus the hit test is performed at a later frame. This causes segmentation masks to be located off in inverse correlation with the camera movement. This issue can be tackled if ARCore frame allows hit testing from an arbitrary pose and direction besides the current location of the camera.

5. EVALUATION

This chapter presents the evaluation results of the semantic image segmentation network and the developed application using ARCore [3]. In the first section of the chapter, we focus on the semantic segmentation performance and efficiency on different challenges in the field. Later, we evaluate the use of scene understanding provided by the proposed network and 3D tracking provided by the ARCore in conjunction.

5.1. Evaluation of the Semantic Segmentation Model

We evaluate our proposed architecture for an FCNN based solution to semantic image segmentation on the PASCAL VOC 2012 and Cityscapes challenge. The following part will discuss some of the current challenges which provide finely annotated datasets that are used in our training and evaluation. Afterward, we will present our training schema for these datasets to obtain our results. Later in the section, we present our results and compare them to the other efficient solutions in the field.

5.1.1. Introduction to the Datasets and Benchmarks

This subsection will provide a brief introduction to the datasets that were used in the evaluation of our proposed model for performing semantic image segmentation.

Common Object Segmentation

The Microsoft Common Objects in Context (COCO) is a data set for detecting and segmenting objects that are found in their natural and everyday life environments. The aim of the data set is to drive the advancement of object detection and segmentation algorithms. The data set presents the non-iconic images of the objects in their ordinal environments and from diverse view angles. Labels in the dataset are annotated per-instance, making the dataset available for instance-aware approaches. In fact, the aim of the challenge on object detection task that is held yearly by COCO is only for instance-aware solutions. Examples of the dataset samples can be seen in Figure 26. [15]

The dataset has a rich selection of object classes and each class has a high number of labeled instances. In the 2017 version of the dataset, there are 118K training, 5K validation and 41K testing samples. It contains 91 common object categories and 82 of these classes have more than 5 000 labeled instances. Overall, the dataset has 2 500 000 labeled instances of objects among 328 000 images (some samples are not yet released as of 2017). In comparison to widely used ImageNet [27] dataset, COCO has less number of classes but each class has more instances of the categories. COCO contains significantly more object instances per image (7,7) compared to ImageNet (3,0) and PASCAL (2,3) which can aid in learning detailed object models. [15]

The challenge is presented for instance-aware solutions but one can use the annotations to develop approaches that are not instance-aware. In fact, many studies

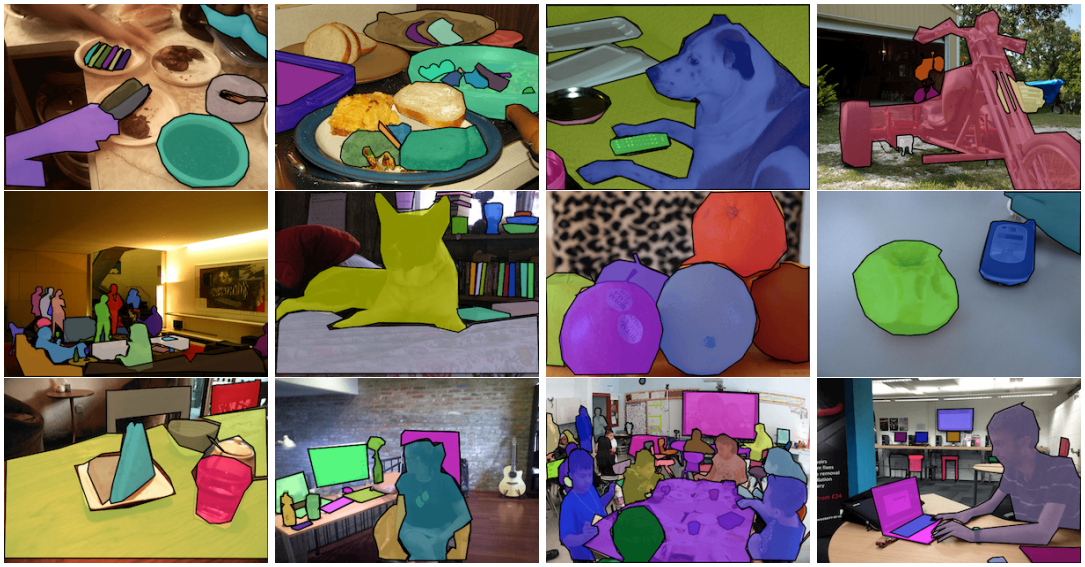


Figure 26. Example images from COCO data set.

use COCO as a supplementary aid in their non-instance-aware tasks such as [30, 1, 2]. In this thesis work, we use COCO for both as a supplementary dataset, and for demonstration of the filtered scene understanding for destructive scene augmentation.

Scene Parsing

This subsection is about the overall introduction to Cityscapes [16], strengths of this urban scene understanding data set and the current state-of-art in the challenge.

Cityscapes is a comprehensive benchmark and a challenge for urban scene understanding. Visual understanding of such urban street scenes has a wide range of applications. Cityscapes consists of a large and diverse set of images where 5 000 of these images are labeled to very high precision and 20 000 additional images have annotated coarsely. Such a large amount of data is crucial for especially learning-based solutions to the problem. Figure 27 shows examples of images and ground truth labels that are in the data set. The dataset aims to spark the progress in semantic urban scene parsing by providing the largest and most distinct dataset of street scenes by having high-quality and coarse annotations, and also presents a sound evaluation methodology for pixel-level and instance level semantic labeling. [16]

Cityscapes is a popular and highly researched challenge in the field. The current state-of-art approaches to the challenge are deep neural network architectures such as DPC [2], RelationNet [67], and DeeplabV3+[1]. In the Cityscapes benchmark, DeeplabV3+ scores $82{,}1\%$ mean intersection over union (mIOU) which was surpassed by DPC, later work of DeeplabV3+, by achieving $82{,}7\%$ mIOU. The DPC and DeeplabV3+ are important architectures for this thesis work as the solution to the scene understanding presented is based upon these approaches.

Figure 27. Example images from Cityscapes data set.

Semantic Segmentation

The Pascal Visual Object Classes (VOC) consists of images taken from Flickr[1] photo sharing website. There are two main challenges presented in VOC: *classification* (does the image contain any occurrence of a specific object class?), and *detection* (where are the occurrence of a specific object class in the image?). Aside from these two challenges, two subsidiary challenge exists: pixel-level segmentation, and person layout localization where the aim is to localize the head, hands, and feet of people in an image. One of the objectives of the challenge is to bring challenging images and high-quality annotation along with a standard evaluation methodology. [17]

The 2012 version of the VOC contains 20 categories among 1 464 train and 1 449 validation samples. However, [68] provides additional annotations for the images in the VOC 2012 dataset which increases training sample count to 10 582. This dataset is often referred as *trainaug* in the literature. Figure 28 shows example images and annotations from the VOC dataset.

We believe that, proving the capability of the solution on variety of classes that are everyday objects, unlike Cityscapes where the primary focus is on the urban scene, shows the possible range applicability of the approach in various fields.

[1] `https://www.flickr.com`

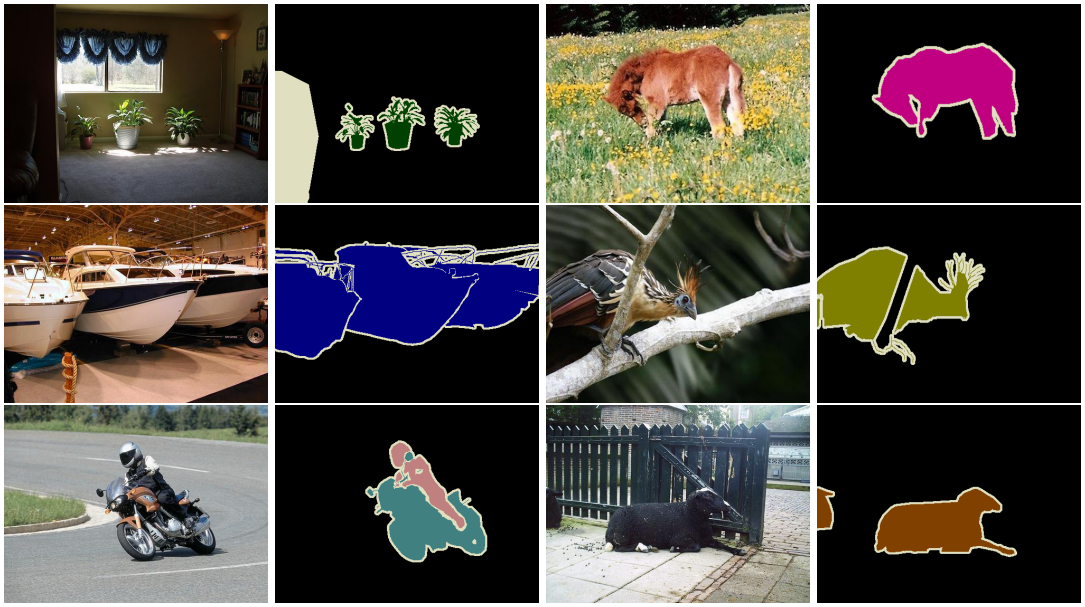


Figure 28. Example images from PASCAL VOC data set.

5.1.2. Training

This subsection describes the training methodology. It starts with the preprocessing of the images and data augmentation and follows by with the training steps that were taken.

Data preprocessing and augmentation

Data preprocessing and augmentation is an important step in the training. For the preprocessing of the images, we standardize the images by

$$inputs \times \frac{2}{255} - 1 \tag{4}$$

to scale the pixel values to $[-1, 1]$ range.

The data augmentation strategy follows the original work of DeeplabV3 and predecessors [1, 30, 2]. Input images are randomly scaled between a factor of $0{,}5$ and $2{,}0$ with steps of $0{,}25$. Then the scaled image is randomly cropped by to the training input dimensions which will be specified for each dataset in the corresponding topics. Lastly, the resulting image cut is randomly flipped on the vertical axis.

Pre-training on COCO

As mentioned earlier in the section, COCO is a data set for detecting and segmenting objects that are found in their natural and everyday life environments. It is a common practice in the field [30, 1, 2] to pre-train the network with COCO dataset to gain accuracy in other challenges such as Cityscapes [16], PASCAL VOC [17] and so on.

The classes that are related to the focus of the challenge should be selected accordingly. Our primary focus is not to evaluate the network on the COCO dataset as it is a challenge on *instance-aware* semantic image segmentation. For that reason,

we choose the classes that are in common with the dataset that we are focusing and combine the instance annotations into a flat class only segmentation annotation. We further limit our image selection to only include the images with annotations of over 1 000 pixels.

Prior to training on COCO, we initially restore the weights of the feature extractor from an ImageNet (ILSVRC) [27] checkpoint. We choose to use large crop sizes and large batch sizes for training. The use of large batch sizes is due to learning *batch normalization* weights as suggested by [30]. The suggested crop size for this step is 513×513 or 769×769 and the suggested batch size should be higher or equal to 16.

Our training was done end-to-end by using a batch size of 16, $output\_stride$ of 16, weight decay set to $4e-5$ to prevent over-fitting, and a "poly" learning rate policy which is formulated as

$$lr^{(k)} = lr_{initial} \left(1 - \frac{k}{max\_iter}\right)^{power} \tag{5}$$

where $lr^{(k)}$ is the learning rate at step k, $lr_{initial}$ set to $0{,}001$ and $power$ set to $0{,}9$ [30] for 60 000 steps using Adam optimizer [69] ($\beta_1 = 0{,}9$, $a_2 = 0{,}999$, $\epsilon = 10^{-8}$). Readers should keep in mind that the training in this step is not fully optimized as COCO is only used for pre-training the network. The loss function is selected as Tensorflow [64] softmax cross entropy loss[2].

Training schema for Cityscapes

As previously mentioned, in the field of computer vision, Cityscapes [16] is a comprehensive benchmark and a challenge for urban scene understanding. Our training schema for the dataset includes restoration from the ImageNet checkpoint, then pre-training according to the above section on person, car, truck, bus, train, motorcycle, bicycle, stop sign and parking meter classes and the 1 000 pixel annotation limit which yields 69 795 training and 2 956 validation samples with a crop size of 769×769.

We further pre-train the network using 20 000 coarsely annotated samples for 60 000 steps, following the same parameters that we used in MS COCO training.

As the final step, we fine-tune the network for 120 000 steps on finely annotated set (2 975 train, 500 validation samples) with an initial learning rate of $0{,}0001$, a slow start at learning rate $1e-5$ for the first 186 steps. We found out that lowering the learning rate at the fine-tuning stage is crucial to achieving good accuracy. Other parameters are kept the same as the previous steps.

Training schema for Pascal VOC 2012

The 2012 version of the VOC [17] contains 20 categories among 1 464 train and 1 449 validation samples. However, [68] provides additional annotations for the images in the VOC 2012 dataset which increases training sample count to 10 582.

The training starts similar to the Cityscapes schema. First, the feature extractor weights are restored from an ImageNet checkpoint. Then, network is pre-trained on

[2] `https://www.tensorflow.org/api_docs/python/tf/losses/softmax_cross_entropy` Tensorflow API r1.13 accessed on 20 May 2019.

COCO dataset samples with the classes that are in common with VOC2012 which are car, bus, bicycle, motorcycle, airplane, boat, train, chair, couch, dining table, tv, bottle, potted plant, cat, dog, cow, horse, sheep, bird, and person. The further filtering was applied similar to the previous where only samples with over 10 000 pixels of annotations in total are selected for training and validation. After this filtering process, COCO yields us 92 516 training and 3 899 validation samples. The network is pre-trained using these samples for 60 000 steps with a base learning rate of $0{,}001$ and a crop size of 513×513. Other parameters were kept the same as the previously mentioned COCO training schema.

Finally, the network is fine-tuned on the PASCAL VOC 2012 *trainaug* set of 69 795 training samples. The initial learning rate for this step was chosen as $0{,}0001$ and the network was trained for 60 000 steps with a crop size of 513×513 and a batch size of 16. All other parameters are kept the same as the previous steps.

5.1.3. Results

This subsection will present the results of the proposed semantic image segmentation solution on Cityscapes and PASCAL VOC.

Cityscapes

Our approach of combining ShuffleNet V2 with DeepLabV3 basic and DPC head achieves both state-of-art mIOU and inference speed with the respective GFLOPs counts. Table 2 shows the comparison of GFLOPs to mIOU performance on the validation set. ShuffleNet V2 with basic DeepLabV3 encoder head, having $2{,}18$ GFLOPs, surpasses the SkipNet-ShuffleNet [33] which has similar floating operations count by $12{,}2\%$ gain on mIOU. On the other hand, DPC variation performs $+0{,}6\%$ more accurate than the MobileNet V2 with basic heads, which has $1{,}54$ times more GFLOPs.

Table 2. Cityscapes **validation** set performance. GFLOPs is measured on image resolution of $640 \times 360 \times 3$.

Method	GFLOPs	mIOU$(\%)$
SkipNet-ShuffleNet [33]	2,0	55,5
ENet [33]	3,83	n/a
MobileNet V2 + Basic	4,69	70,7
MobileNet V2 + DPC	5,56	n/a
ShuffleNet V2 + Basic (Ours)	2,18	67,7
ShuffleNet V2 + DPC (Ours)	3,05	71,3

ShuffleNet V2 with DPC heads outperforms state-of-art efficient networks on the Cityscapes test set. Table 3 shows that we make a gain of 7.27% mIOU over the best performing ENet architecture. Furthermore, on each of the classes highlighted here, ShuffleNet V2+DPC shows a great improvement. Also, as seen in the Table 4, our

approach outperforms the previous methods on the category mIOU and instance level metrics.

Table 3. Comparison of class and category level accuracy on the test set.

Method	mIOU	Building	Sky	Car	Sign	Road	Person	Fence	Pole	Sidewalk	Bicycle
SkipNet-MobileNet	61,52	86,19	92,89	89,88	54,34	95,82	69,25	39,40	44,53	73,93	58,15
Enet Lovasz	63,06	87,22	92,74	91,01	58,06	97,27	71,35	38,99	48,53	77,20	59,80
ShuffleNet V2+DPC(Ours)	70,33	90,7	93,86	93,95	66,93	98,11	78,47	50,93	51,47	82,46	67,48

Table 4. Comparison of class level accuracies efficient architectures on Cityscapes **test** set.

Method	Class IOU	Class iIOU	Cat. IOU	Cat. iIOU
SegNet [42]	56,1	34,2	79,8	66,4
ShuffleSeg [32]	58,3	32,4	80,2	62,6
SkipNet-MobileNet [33]	61,52	35,16	82,00	63,03
Enet Lovasz [31]	63,06	34,06	83,58	61,05
ShuffleNet V2+DPC (Ours)	70,33	43,58	86,48	69,92

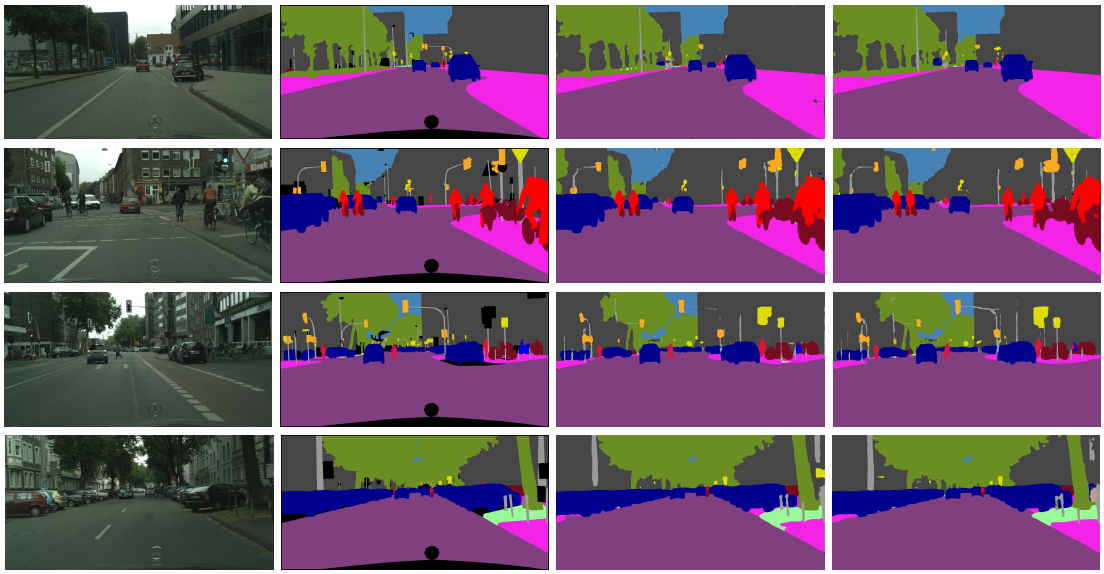
We visualize the segmentation masks generated both by the basic and DPC approach in Figure 29. From the visuals, we can see that the DPC head variant is able to segment thinner objects such as poles better than the basic encoder head. The figure is best viewed in color.

PASCAL VOC

Our proposed model performs poorly compared to MobileNet V2 [19]. As seen in Table 5, accuracy wise, the ShuffleNet V2 backbone falls behind the accuracy of the MobileNet V2. One possible reason for the comparison might be that the training hyper-parameters were not fully discovered during the training on PASCAL VOC 2012. Although the accuracy is lower than the MobileNet V2 architecture, when compared in respect to FLOPs, our model, having $2{,}43$ GFLOPs, operates at $53{,}6\%$ fewer FLOPs over its counterpart.

Table 5. PASCAL VOC 2012 **validation** set results. GFLOPs is measured on image resolution of $512 \times 512 \times 3$.

Method	Output Stride	GFLOPs	mIOU (%)
MobileNet V2 [19]	16	5,24	75,32
ShuffleNet V2 + Basic	16	2,43	65,90
ShuffleNet V2 + DPC	16	3,41	67,05



(a) Input image (b) Ground truth (c) Basic head (d) DPC

Figure 29. Segmentation visualizations on the Cityscapes validation set. Best viewed in color. (Black colored regions on ground truth are ignored)

5.1.4. Performance on Mobile Device

This section provides inference speed results on the proposed semantic image segmentation network. We believe that computational efficiency is best when it is demonstrated on the actual device by measuring the inference time rather than comparing the FLOPs count as discussed in [21]. To this extent, we converted the competing networks to Tensorflow Lite [65], a binary model representation for inferencing on mobile phones by Tensorflow [64]. The number of available operations that are supported by Tensorflow Lite is limited in its current implementation, thus complex architectures such as [70] cannot be converted without implementing the missing operations on the Tensorflow Lite. But, both MobileNet V2 and ShuffleNet V2 are compatible and require no additional custom operations.

The inference speed analysis was performed on OnePlus A6003 (Snapdragon 845 CPU, 6GB RAM, 16 + 20 MP Dual Camera, Android version 9, OxygenOS version 9.0.3). The device is stabilized in place, put to airplane mode, background services are disabled, the battery is fully charged and kept plugged to the power cable. The rear camera output image is scaled so that the smaller dimension is 224, then the middle of the image is cropped to get an input image of 224×224. The downscaling and cropping is not included in the inference time shown in Table 6 but the preprocessing of the input image, according to Equation 4, and the final ArgMax is included in the inference time. Inference speed measurements, presented in Table 6, are averaged over 300 frames after 30 seconds of an initial warm-up period.

Table 6 shows that our approach of using ShuffleNet V2 as a backbone is capable of performing real-time semantic segmentation, close to 20Hz, on a mobile phone. Even with the complex DPC architecture, ShuffleNet V2 backbone ends up with $1{,}54$ times fewer GFLOPs over the MobileNet V2 with basic encoder head. Furthermore, model size is significantly lower which is desired for embedded devices that might

Table 6. Inference performance on OnePlus A6 with an input size of 224×224 and class count of 21 (Cityscapes).

Backbone	Encoder	GFLOPs	Inference (ms)	Var (ms^2)	FPS	Size (MB)
ShuffleNet V2	Basic	0.47	50.89	0.57	19.65	4.6
	DPC	0.65	64.89	3.53	15.41	6
MobileNet V2	Basic	1.00	101.46	25.04	9.86	8.4
	DPC	1.18	116.16	44.01	8.61	9.9

have limited memory or storage size. One surprising finding is that the MobileNet V2 variants show much higher variance in the inference time compared to the ShuffleNet V2 backbone, thus our approach provides a more stable fps.

5.2. Evaluation of the tracking

This section now turns to the evaluation of Android-based augmented reality application that takes advantage of the scene understanding provided by the semantic image segmentation. The developed application is evaluated quantitatively by recording the application while it is running on the device and then each frame is labeled by hand to acquire the ground truth labels. This data is then used for measuring mIOU and the stability of the system when used together with 3D tracking provided by ARCore [3].

Methodology

To evaluate the accuracy of the tracking and masking that is performed by the application, we use Huawei Mate 20 Pro LYA-L29 (Huawei Kirin 980 CPU, 6GB RAM, 40 MP Wide Angle Lens + 20 MP Ultra-Wide Angle Lens + 8 MP Telephoto triple camera, Android version 9, EMUI 9.0.0) and take the following steps:

1. The phone is fully charged, put to airplane mode and the application is kept open for 30-60 seconds for warmup.
2. The screen of the device is recorded on the phone for 20 seconds for each case, where the first case is slow movements and the second case is fast movements.
3. The video files are then processed by hand by labeling each frame and also outlining the on-device segmentation masks.
4. The data that was collected and labeled is evaluated in two aspects.
   (a) Mean intersection over union with regards to the ground truth labels.
   (b) The steadiness of the segmentation mask.
5. Additional data is collected for insights on battery consumptions at different segmentation mask calculation rate.

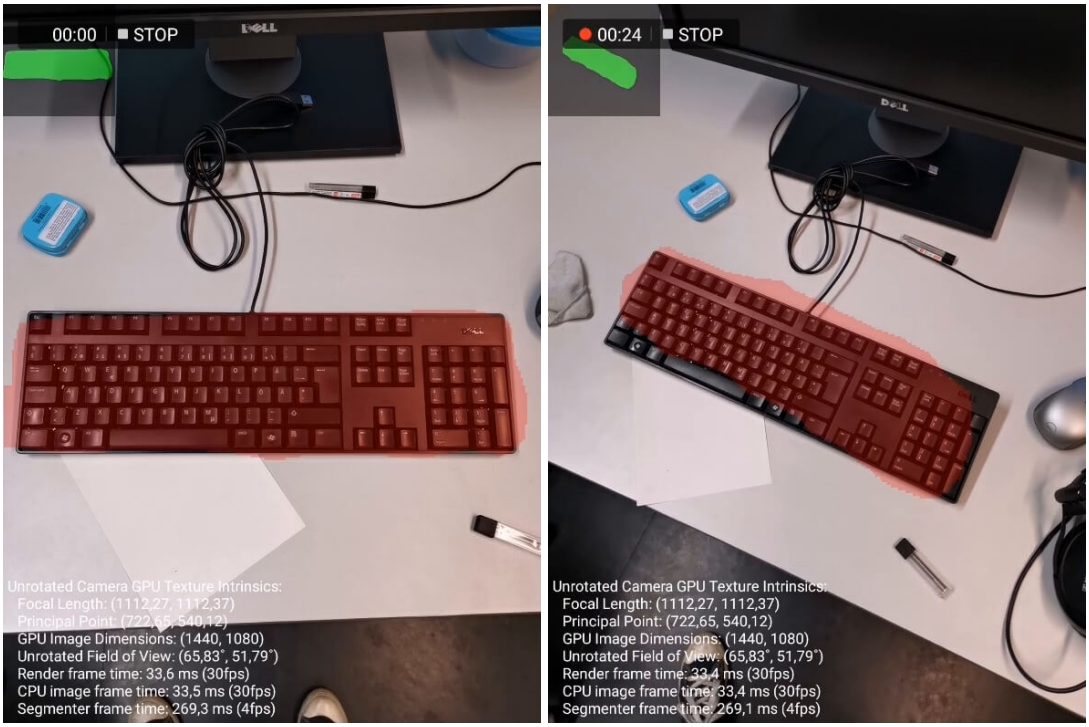


Figure 30. Example images of the testing environment.

Although the DPC head offers higher accuracy, the evaluation used ShuffleNet V2 with basic head operating at image dimensions of 225×225 and output stride of 16 for the evaluation on the device due to its faster inference time. The trained model had four classes which were apple, keyboard, book, and background. For the demonstration, we choose to only use keyboard class.

The environment of the test consisted of several items on a single desk. The items including a keyboard were placed as a normal desk and keyboard would be placed as seen in Figure 30. The objects in the scene were kept stationary throughout the data collection. The room was also bright enough to ease the plane detection and semantic segmentation.

After an initial warm-up and plane detection, three separate videos were recorded. The first video consisted of slow movements of the handheld phone. The second video consisted of fast changes in the camera location and direction and lastly, in the third video, the online segmentation was paused after an initial scan of the object. All three videos were 20 seconds long and from these videos, 4 images per each second were extracted which results in 300 frames in total. Each extracted frame was carefully annotated for both the ground truth and the prediction masks. These masks are then evaluated for mIOU and centroid error analysis of the overall system as seen in Table 7.

Results

The third video, where the segmentation was paused, aims to capture the accuracy of the tracking while the other two videos aim to measure the accuracy of the whole system.

Table 7. Accuracy result of the 3D tracked segmentation masks. Centroid denotes the centroid of the ground truth labels and error denotes the distance of the predicted segmentation mask centroid. Values are absolute pixel distances on 590×789 images.

Method	Online Segmentation	Segmentation Period (ms)	Centroid (*std*) x	Centroid (*std*) y	RMSE	mIOU (%)
Tracking Only[1]	✗	–	82,28	84,25	4,73	93,85
Slow Movement	✓	250	16,10	50,80	7,94	90,55
Fast Movement	✓	250	40,76	65,90	17,09	83,48

[1] Initial mask IOU score is $96{,}06\%$.

The tracking only methodology seems to accurately follow and update the position of the mask. As seen in Table 7, the overall mIOU was measured as $93{,}85\%$ after the initial scan of the object which scored $96{,}06\%$ despite the heavy movements of the device. Furthermore, we see that the centroid of the segmentation mask was kept very steady at root mean squared error (RMSE) of $4{,}73$ pixels in an image with 590×789 dimensions. The centroid standard deviation (STD) of the ground truth gives an insight into the magnitude of the movements that were performed during the recording, but, it should be noted that STD of *x* and *y* does not fully reflect the movements that were performed around Z-axis since they are 2D projections of a 3D point.

We see that overall performance on the online segmentation of the object and the tracking has achieved a high mIOU, $90{,}55$ for slow movements and $83{,}48$ for fast movements, but we see a significant drop on the fast movement case as shown in Table 7. The standard deviation of the ground truth centroids shows that we successfully captured our desired 'slow' and 'fast' cases. Furthermore, errors on the centroid of the predicted labels in the fast movement case indicate that the third limitation that was listed in Section 4.4 is an effect in the localization of the segmentation mask. This could be tackled if the ARCore framework allows hit testing on previous frames where the image was obtained for semantic segmentation. That way the segmentation map should be always placed as in the slow movement case.

Our results show that running semantic image segmentation for continuous scene understanding increases for battery consumption. Although it is not a direct metric, the CPU usage of the application is an indication of the battery consumption level of an application. To test the battery effect of the system we experiment on our testing smartphone, Huawei Mate 20 Pro, using segmentation at frequencies of segmentation requests. The application CPU usage was collected for 100 seconds at 1-second intervals after an initial warmup around 10-30 seconds. Table 8 shows that the more frequent the segmentation, the more CPU is used by the application. Looking at the plot in Figure 31, we see that the trend is similar to a linear growth as we use higher fps and it reaches a limit near 100% as the network is running on single core of the CPU. Thus, to obtain scene understanding, one can exploit the 3D tracking offered by ARCore to lower the energy consumption level by operating at longer intervals of semantic image segmentation.

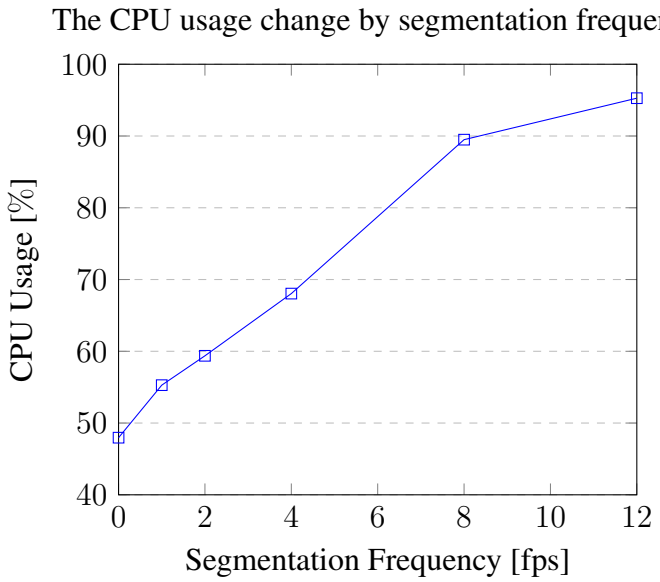


Figure 31. The CPU usage by requests of semantic segmentation at different frequencies.

Table 8. The CPU usage of the application for different periods of segmentation inference. The maximum percentage of CPU usage is 800% on the reference device Huawei Mate 20 Pro.

Segmentation Period (ms)	CPU usage (%) *mean*	*std*
83	95,27	5,99
125	89,49	3,62
250	68,04	2,94
500	59,37	4,00
1000	55,28	3,36
No Segment	47,96	1,84

5.3. In-painting objects as a possible application

A possible application of using semantic segmentation along with ARCore is to remove some objects from the real-world environment. This is a type of destructive augmentation. There are also other possible applications such as highlighting or outlining the target objects in the view, darkening the background to collect the focus on the objective and so on.

As discussed in the implementation, our proposed system is an example of inpainting objects. inpainting is a challenging task especially if there are big areas as a target. The implementation in this thesis work is intended as a showcase rather than aiming for an accurate inpainting, thus we demonstrate the results in Figure 32.

Although our implementation does not provide an appealing inpainting result, since the segmentation masks give the interest pixels, better solutions can be implemented as future work. Furthermore, ARCore offers a light estimation of the current environment.

Figure 32. Inpainted keyboards on different surfaces.

This information can be added to the inpainting algorithm to achieve realistic looking inpainting results.

6. DISCUSSION

In this chapter, we present the strengths and limitations of the thesis work on the semantic image segmentation model that was proposed as an efficient method to scene understanding and the joint application of 3D tracking of the masks that were generated by the network using ARCore augmented reality framework.

A key strength of this thesis is the proposed deep neural network architecture for efficient semantic image segmentation. The model achieved a state-of-art performance on balancing efficiency and accuracy. In this extent, two separate models were demonstrated. The first, ShuffleNet V2 + Basic had the simplicity and efficiency in the foreground while the second, ShuffleNet V2 + DPC had a focus on improving the accuracy while keeping the efficiency. The findings on this topic were published as a paper [63] in SCIA 2019 [71]. The semantic information offered by the solution has many applications such as autonomous driving, robotic navigation, localization, and scene understanding where the resources may be limited. Specifically, for augmented reality, it is possible to use this information for many applications such as removing objects from the scene, collecting attention to the object in focus, providing scene understanding to the logic and so on. In future investigations, it might be possible to use a U-Net [34] style decoder with skip connections from earlier stages of the network to obtain sharper edges around the object boundaries. Furthermore, the architecture of the ShuffleNet V2 is designed for ILSVRC [27] challenge where there are 1000 classes to predict, thus, there might be room for squeezing the channel count of the lower stages of the network for an additional boost in efficiency while keeping the similar accuracy.

In addition, the destructive augmentation developed in this thesis work, despite its current limitations, offers some insight into the utilization of the semantic information of the current context. Augmented reality is an emerging field that is starting to appear in daily life. One of the most important use cases of augmented reality is to aid the operator to increase performance. Since the pre-modeled environment approach for scene understanding in AR lacks the support for non-static objects, semantic image segmentation fills the gap by providing online information. The implementation of our system also shows that for specific cases such as inpainting objects, continues evaluation of the context is not necessary to keep the model updated. The objects that were discovered can be tracked in 3D thanks to the inertial-visual odometry, thus the frequency of the discovery can be lowered for saving the battery and computational power. Current demonstration limits the use to fairly flat objects such as keyboards, thus, further work in supporting the height of the objects is possible. This might be solved by obtaining a 3D map of the context through the structure from motion to estimate the object shape. In addition, to adjust for the semantic understanding delay, hit testing can be performed by using the location and the direction of the device at the instant of the captured image that was selected for processing.

There is abundant room for further progress in the inpainting methodology that was used in this thesis work. Our naive approach is far from providing a natural looking overlay of the objects. Algorithms such as [72] are promising possibilities but the efficiency of the methodology should be kept in mind as the nature of the system in our focus is a real-time system. Furthermore, another big challenge is to achieve

deterministic inpainting that will not Flicker or change the resulting scene drastically over the movements of the camera and changes in the environment.

7. CONCLUSION

This thesis set out to develop a model for scene understanding in augmented reality on everyday devices such as smartphones. To this extent, two main topics, learning-based semantic image segmentation, and 3D tracking, were explored. The methodology that was developed used semantic image segmentation for scene understanding while tracking the segmentation masks in 3D real-world environment and allowing further augmentation of the environment through the use of ARCore. Furthermore, an example of the usage of the semantic information was demonstrated that aims to hide objects to perform destructive augmentation on an Android smartphone. Since the computational power of such devices is scarce, the neural network for the semantic image segmentation was designed computationally efficient to achieve low latency. The overall system is shown to have a high mIOU at various segmentation frequencies, and also, at only-tracking case after an initial segmentation. In addition, tracking the segmentation mask rather than calculating each frame seems to lower the CPU usage of the application, thus decreasing the battery usage.

8. REFERENCES

[1] Chen L., Zhu Y., Papandreou G., Schroff F. & Adam H. (2018) Encoder-decoder with atrous separable convolution for semantic image segmentation. In: V. Ferrari, M. Hebert, C. Sminchisescu & Y. Weiss (eds.) Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part VII, Lecture Notes in Computer Science, vol. 11211, Springer, Lecture Notes in Computer Science, vol. 11211, pp. 833–851. URL: https://doi.org/10.1007/978-3-030-01234-2_49.

[2] Chen L., Collins M.D., Zhu Y., Papandreou G., Zoph B., Schroff F., Adam H. & Shlens J. (2018) Searching for efficient multi-scale architectures for dense image prediction. In: S. Bengio, H.M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi & R. Garnett (eds.) Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada., pp. 8713–8724. URL: http://papers.nips.cc/paper/8087-searching-for-efficient-multi-scale-architectures-for-dense-image-prediction.

[3] Google LLC, ARCore by Google. URL: https://play.google.com/store/apps/details?id=com.google.ar.core, (Accessed: 2019-04-08).

[4] Rosenberg L.B. (1992) The use of virtual fixtures as perceptual overlays to enhance operator performance in remote environments. Tech. rep., Stanford Univ Ca Center for Design Research.

[5] Azuma R.T. (1997) A survey of augmented reality. Presence: Teleoperators and Virtual Environments 6, pp. 355–385.

[6] Eve S. (2012) Augmenting phenomenology: Using augmented reality to aid archaeological phenomenology in the landscape. Journal of Archaeological Method and Theory 19, pp. 582–600.

[7] Chi H.L., Kang S.C. & Wang X. (2013) Research trends and opportunities of augmented reality applications in architecture, engineering, and construction. Automation in Construction 33, pp. 116–122.

[8] Greg Kipper J.R. (2012) Augmented Reality: An Emerging Technologies Guide to AR. SYNGRESS MEDIA. URL: https://www.ebook.de/de/product/18450125/greg_kipper_joseph_rampolla_augmented_reality_an_emerging_technologies_guide_to_ar.html.

[9] Zhu W., Owen C.B., Li H. & hyun Lee J. (2004) Personalized in-store e-commerce with the promopad: an augmented reality shopping assistant. Electronic Journal for E-commerce Tools and Applications 1. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.83.8198.

[10] Wu H., Lee S.W., Chang H. & Liang J. (2013) Current status, opportunities and challenges of augmented reality in education. Computers & Education 62, pp. 41–49.

[11] Houzz Inc., Houzz. URL: https://itunes.apple.com/us/app/houzz/id399563465, (Accessed: 2019-04-04).

[12] Inter IKEA Systems B.V., IKEA Place. URL: https://itunes.apple.com/us/app/ikea-place/id1279244498, (Accessed: 2019-05-15).

[13] Shelhamer E., Long J. & Darrell T. (2017) Fully convolutional networks for semantic segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence 39, p. 640–651. URL: http://dx.doi.org/10.1109/TPAMI.2016.2572683.

[14] Zhou B., Zhao H., Puig X., Xiao T., Fidler S., Barriuso A. & Torralba A. (2018) Semantic understanding of scenes through the ade20k dataset. International Journal of Computer Vision 127, p. 302–321. URL: http://dx.doi.org/10.1007/s11263-018-1140-0.

[15] Lin T., Maire M., Belongie S.J., Hays J., Perona P., Ramanan D., Dollár P. & Zitnick C.L. (2014) Microsoft COCO: common objects in context. In: D.J. Fleet, T. Pajdla, B. Schiele & T. Tuytelaars (eds.) Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V, Lecture Notes in Computer Science, vol. 8693, Springer, Lecture Notes in Computer Science, vol. 8693, pp. 740–755. URL: https://doi.org/10.1007/978-3-319-10602-1_48.

[16] Cordts M., Omran M., Ramos S., Rehfeld T., Enzweiler M., Benenson R., Franke U., Roth S. & Schiele B. (2016) The cityscapes dataset for semantic urban scene understanding. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016, IEEE Computer Society, pp. 3213–3223.

[17] Everingham M., Eslami S.M.A., Van Gool L., Williams C.K.I., Winn J. & Zisserman A. (2015) The pascal visual object classes challenge: A retrospective. International Journal of Computer Vision 111, pp. 98–136. URL: https://doi.org/10.1007/s11263-014-0733-5.

[18] Howard A.G., Zhu M., Chen B., Kalenichenko D., Wang W., Weyand T., Andreetto M. & Adam H. (2017) Mobilenets: Efficient convolutional neural networks for mobile vision applications. CoRR abs/1704.04861. URL: http://arxiv.org/abs/1704.04861.

[19] Sandler M., Howard A., Zhu M., Zhmoginov A. & Chen L.C. (2018) Mobilenetv2: Inverted residuals and linear bottlenecks. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition URL: http://dx.doi.org/10.1109/CVPR.2018.00474.

[20] Zhang X., Zhou X., Lin M. & Sun J. (2018) Shufflenet: An extremely efficient convolutional neural network for mobile devices. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition URL: http://dx.doi.org/10.1109/cvpr.2018.00716.

[21] Ma N., Zhang X., Zheng H.T. & Sun J. (2018) Shufflenet v2: Practical guidelines for efficient cnn architecture design. Lecture Notes in Computer Science , p. 122–138URL: http://dx.doi.org/10.1007/978-3-030-01264-9_8.

[22] Zhang Y.J. (2006) An overview of image and video segmentation in the last 40 years. In: Advances in Image and Video Segmentation, IGI Global, pp. 1–16.

[23] Shapiro L.G. & Stockman G.C. (2001) Computer Vision. Pearson. URL: https://www.amazon.com/Computer-Vision-Linda-G-Shapiro/dp/0130307963.

[24] Roberts L.G. (1963) Machine Perception of Three-Dimensional Solids. Outstanding Dissertations in the Computer Sciences, Garland Publishing, New York.

[25] LeCun Y., Bengio Y. & Hinton G.E. (2015) Deep learning. Nature 521, pp. 436–444.

[26] Krizhevsky A., Sutskever I. & Hinton G.E. (2012) Imagenet classification with deep convolutional neural networks. In: P.L. Bartlett, F.C.N. Pereira, C.J.C. Burges, L. Bottou & K.Q. Weinberger (eds.) Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States., pp. 1106–1114. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.

[27] Russakovsky O., Deng J., Su H., Krause J., Satheesh S., Ma S., Huang Z., Karpathy A., Khosla A., Bernstein M. & et al. (2015) Imagenet large scale visual recognition challenge. International Journal of Computer Vision 115, p. 211–252. URL: http://dx.doi.org/10.1007/s11263-015-0816-y.

[28] Deng L. (2012) The MNIST database of handwritten digit images for machine learning research [best of the web]. IEEE Signal Process. Mag. 29, pp. 141–142.

[29] Hinton G., Deng L., Yu D., Dahl G., rahman Mohamed A., Jaitly N., Senior A., Vanhoucke V., Nguyen P., Sainath T. & Kingsbury B. (2012) Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. IEEE Signal Processing Magazine 29, pp. 82–97.

[30] Chen L., Papandreou G., Schroff F. & Adam H. (2017) Rethinking atrous convolution for semantic image segmentation. CoRR abs/1706.05587. URL: http://arxiv.org/abs/1706.05587.

[31] Berman M., Triki A.R. & Blaschko M.B. (2018) The lovasz-softmax loss: A tractable surrogate for the optimization of the intersection-over-union measure in neural networks. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition URL: http://dx.doi.org/10.1109/CVPR.2018.00464.

[32] Gamal M., Siam M. & Abdel-Razek M. (2018) Shuffleseg: Real-time semantic segmentation network. CoRR abs/1803.03816. URL: http://arxiv.org/abs/1803.03816.

[33] Siam M., Gamal M., Abdel-Razek M., Yogamani S., Jägersand M. & Zhang H. (2018) A comparative study of real-time semantic segmentation for autonomous driving. In: 2018 IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2018, Salt Lake City, UT, USA, June 18-22, 2018, IEEE Computer Society, pp. 587–597. URL: http://openaccess.thecvf.com/content_cvpr_2018_workshops/w12/html/Siam_A_Comparative_Study_CVPR_2018_paper.html.

[34] Paszke A., Chaurasia A., Kim S. & Culurciello E. (2016) Enet: A deep neural network architecture for real-time semantic segmentation. CoRR abs/1606.02147. URL: http://arxiv.org/abs/1606.02147.

[35] Glorot X., Bordes A. & Bengio Y. (2011) Deep sparse rectifier neural networks. In: G.J. Gordon, D.B. Dunson & M. Dudík (eds.) Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011, JMLR Proceedings, vol. 15, JMLR.org, JMLR Proceedings, vol. 15, pp. 315–323. URL: http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf.

[36] Goodfellow I.J., Bengio Y. & Courville A.C. (2016) Deep Learning. Adaptive computation and machine learning, MIT Press. URL: http://www.deeplearningbook.org/.

[37] Yu F. & Koltun V. (2016) Multi-scale context aggregation by dilated convolutions. In: Y. Bengio & Y. LeCun (eds.) 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings. URL: http://arxiv.org/abs/1511.07122.

[38] Ioffe S. & Szegedy C. (2015) Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: F.R. Bach & D.M. Blei (eds.) Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015, JMLR Workshop and Conference Proceedings, vol. 37, JMLR.org, JMLR Workshop and Conference Proceedings, vol. 37, pp. 448–456. URL: http://jmlr.org/proceedings/papers/v37/ioffe15.html.

[39] Meyer-Baese A. & Schmid V. (2014) Feature selection and extraction. In: Pattern Recognition and Signal Analysis in Medical Imaging, Elsevier, pp. 21–69.

[40] Chollet F. (2017) Xception: Deep learning with depthwise separable convolutions. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017, IEEE Computer Society, pp. 1800–1807.

[41] Ronneberger O., Fischer P. & Brox T. (2015) U-Net: Convolutional networks for biomedical image segmentation. In: Lecture Notes in Computer Science, Springer International Publishing, pp. 234–241.

[42] Badrinarayanan V., Kendall A. & Cipolla R. (2017) Segnet: A deep convolutional encoder-decoder architecture for image segmentation. IEEE Trans. Pattern Anal. Mach. Intell. 39, pp. 2481–2495.

[43] Bergstra J. & Bengio Y. (2012) Random search for hyper-parameter optimization. Journal of Machine Learning Research 13, pp. 281–305. URL: http://dl.acm.org/citation.cfm?id=2188395.

[44] Caudell T.P. & Mizell D.W. (1992) Augmented reality: an application of heads-up display technology to manual manufacturing processes. In: Proc. Twenty-Fifth Hawaii Int. Conf. System Sciences, vol. ii, vol. ii, pp. 659–669 vol.2.

[45] Sutherland I.E. (1968) A head-mounted three dimensional display. In: American Federation of Information Processing Societies: Proceedings of the AFIPS '68 Fall Joint Computer Conference, December 9-11, 1968, San Francisco, California, USA - Part I, AFIPS Conference Proceedings, vol. 33, AFIPS / ACM / Thomson Book Company, Washington D.C., AFIPS Conference Proceedings, vol. 33, pp. 757–764.

[46] Augmented Reality iOS. URL: https://www.apple.com/ios/augmented-reality/, (Accessed: 2019-05-09).

[47] Milgram P., Takemura H., Utsumi A. & Kishino F. (1995) Augmented reality: a class of displays on the reality-virtuality continuum. In: H. Das (ed.) Telemanipulator and Telepresence Technologies, vol. 2351, International Society for Optics and Photonics, SPIE, vol. 2351, pp. 282–293.

[48] Krueger M.W., Gionfriddo T. & Hinrichsen K. (1985) VIDEOPLACE—an artificial reality. In: Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '85, vol. 16, ACM, ACM Press, vol. 16, pp. 35–40.

[49] Azuma R., Baillot Y., Behringer R., Feiner S., Julier S. & MacIntyre B. (2001) Recent advances in augmented reality. IEEE Computer Graphics and Applications 21, pp. 34–47.

[50] Google Glass. URL: https://www.google.com/glass/start/, (Accessed: 2019-06-17).

[51] Microsoft HoloLens: Mixed Reality Technology for Business. URL: https://www.microsoft.com/en-us/hololens/, (Accessed: 2019-05-11).

[52] Wagner D., Reitmayr G., Mulloni A., Drummond T. & Schmalstieg D. (2010) Real-time detection and tracking for augmented reality on mobile phones. IEEE Trans. Vis. Comput. Graph. 16, pp. 355–368.

[53] Bajura M. & Neumann U. (1995) Dynamic registration correction in augmented-reality systems. In: 1995 Virtual Reality Annual International Symposium, VRAIS '95, Research Triangle Park, North Carolina, USA, March 11-15, 1995, IEEE Computer Society, pp. 189–197.

[54] Schmalstieg D. & Höllerer T. (2016) Augmented reality: principles and practice. In: Special Interest Group on Computer Graphics and Interactive Techniques Conference, SIGGRAPH '16, Anaheim, CA, USA, July 24-28, 2016, Courses, ACM, p. 6:1.

[55] Durrant-Whyte H.F. & Bailey T. (2006) Simultaneous localization and mapping: part I. IEEE Robot. Automat. Mag. 13, pp. 99–110.

[56] Harris C.G. & Stephens M. (1988) A combined corner and edge detector. In: C.J. Taylor (ed.) Proceedings of the Alvey Vision Conference, AVC 1988, Manchester, UK, September, 1988, Alvey Vision Club, pp. 1–6.

[57] Ojala T., Pietikäinen M. & Mäenpää T. (2002) Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. IEEE Trans. Pattern Anal. Mach. Intell. 24, pp. 971–987.

[58] Lucas B.D. & Kanade T. (1981) An iterative image registration technique with an application to stereo vision. In: Proceedings of the 7th International Joint Conference on Artificial Intelligence, IJCAI '81, Vancouver, BC, Canada, August 24-28, 1981, pp. 674–679.

[59] Nistér D. (2004) An efficient solution to the five-point relative pose problem. IEEE Trans. Pattern Anal. Mach. Intell. 26, pp. 756–777.

[60] Fischler M.A. & Bolles R.C. (1981) Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM 24, pp. 381–395.

[61] Leutenegger S., Lynen S., Bosse M., Siegwart R. & Furgale P.T. (2015) Keyframe-based visual-inertial odometry using nonlinear optimization. I. J. Robotics Res. 34, pp. 314–334.

[62] van Krevelen D.W.F. & Poelman R. (2010) A survey of augmented reality technologies, applications and limitations. International Journal of Virtual Reality 9, pp. 1–20. URL: http://www.ijvr.org/issues/issue2-2010/paper1%20.pdf.

[63] Türkmen S. & Heikkilä J. (2019) An efficient solution for semantic segmentation: ShuffleNet v2 with atrous separable convolutions. In: Image Analysis, Springer International Publishing, pp. 41–53. URL: http://dx.doi.org/10.1007/978-3-030-20205-7_4.

[64] Abadi M., Barham P., Chen J., Chen Z., Davis A., Dean J., Devin M., Ghemawat S., Irving G., Isard M., Kudlur M., Levenberg J., Monga R., Moore S., Murray D.G., Steiner B., Tucker P., Vasudevan V., Warden P., Wicke M., Yu Y. & Zheng X. (2016) Tensorflow: A system for large-scale machine learning. In: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), pp. 265–283. URL: https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf.

[65] TensorFlow Lite. URL: https://www.tensorflow.org/lite/, (Accessed: 2019-05-11).

[66] Bradski G. (2000) The OpenCV Library. Dr. Dobb's Journal of Software Tools .

[67] Zhuang Y., Tao L., Yang F., Ma C., Zhang Z., Jia H. & Xie X. (2018) Relationnet: Learning deep-aligned representation for semantic image segmentation. In: 24th International Conference on Pattern Recognition, ICPR 2018, Beijing, China, August 20-24, 2018, IEEE Computer Society, pp. 1506–1511.

[68] Hariharan B., Arbelaez P., Bourdev L.D., Maji S. & Malik J. (2011) Semantic contours from inverse detectors. In: D.N. Metaxas, L. Quan, A. Sanfeliu & L.J.V. Gool (eds.) IEEE International Conference on Computer Vision, ICCV 2011, Barcelona, Spain, November 6-13, 2011, IEEE Computer Society, pp. 991–998.

[69] Kingma D.P. & Ba J. (2015) Adam: A method for stochastic optimization. In: Y. Bengio & Y. LeCun (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings. URL: http://arxiv.org/abs/1412.6980.

[70] He K., Gkioxari G., Dollár P. & Girshick R.B. (2017) Mask R-CNN. In: IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017, IEEE Computer Society, pp. 2980–2988.

[71] Felsberg M., Forssén P.E., Sintorn I.M. & Unger J. (eds.) (2019) Image Analysis. Springer International Publishing. URL: https://dx.doi.org/10.1007/978-3-030-20205-7.

[72] He K. & Sun J. (2012) Statistics of patch offsets for image completion. In: Computer Vision – ECCV 2012, Springer Berlin Heidelberg, pp. 16–29.