# Semantic Understanding for Augmented Reality and Its Applications

Thesis presented on April 8, 2020

JOSEPH DECHICCHIS, Duke University, Department of Computer Science

**Abstract:** Although augmented reality (AR) devices and developer toolkits are becoming increasingly ubiquitous, current AR devices lack a semantic understanding of the user's environment. Semantic understanding in an AR context is critical to improving the AR experience because it aids in narrowing the gap between the physical and virtual worlds, making AR more seamless as virtual content interacts naturally with the physical environment. A granular understanding of the user's environment has the potential to be applied to a wide variety of problems, such as visual output security, improved mesh generation, and semantic map building of the world. This project investigates semantic understanding for AR by building and deploying a system which uses a semantic segmentation model and Magic Leap One to bring semantic understanding to a physical AR device, and explores applications of semantic understanding such as visual output security using reinforcement learning trained policies and the use of semantic context to improve mesh quality.

**Committee members:**
Maria Gorlatova (advisor)
Daniel J. Sorin
Carlo Tomasi

Thesis presented in partial fulfillment of the requirements for graduating with distinction in the Department of Computer Science at Duke University
Spring 2020

## Contents

## 1 INTRODUCTION

Augmented reality (AR) technology is becoming increasingly ubiquitous due to the proliferation of mobile devices and developer toolkits such as ARKit [4] and ARCore [22] as well as the introduction of head mounted AR devices such as the HoloLens [43] and the Magic Leap One [36] which enable rich and immersive AR experiences. Furthermore, we can expect consumer head mounted AR devices to enter the market as hardware improves, form factors become more portable and comfortable, and unit costs decrease. It has been reported that Apple is working on consumer AR glasses [71] and they are most definitely not the only consumer technology company working to bring AR to the masses.

Even though AR applications are available to consumers, there is still much to be desired in terms of a truly immersive and rich experience, even on dedicated AR devices such as the Magic Leap One. Ideal hardware improvements for future generation devices include: a smaller and lighter form factor, larger field of view, better color reproduction, and higher resolution displays. While these hardware improvements will surely be welcome by users of AR devices, software needs to improve as well. In particular, in order for AR applications to not only overlay content in front of the user's field of view but also enable users to *interact* with digital content requires software improvements in several areas, such as better simultaneous localization and mapping (SLAM), semantic scene understanding, and multi-user AR. A seamless interaction between the physical and virtual worlds, sometimes referred to as Mixed Reality [44], would allow for a much more immersive AR experience.

Current AR devices already have a limited understanding of the user's environment. For example, the Magic Leap One is able to create a mesh of the user's environment and localize the user within this mesh. The device can also store meshes and share them with the cloud. However, meshes only give AR applications a sense of where surfaces are, without any information on what that surface is (e.g. table, wall, floor etc.). Nonetheless, specific algorithms have been developed for mobile AR to, for example, understand the detailed features of a user's face. Although such technology is used by Snapchat [66] and Facebook Messenger [19] to overlay a user's face with various artificial masks, it does not provide a more general understanding of the user's environment. While Apple introduced People Occlusion to ARKit [4], it is still limited to only detecting occlusion for people in a scene and cannot detect occlusion for any object.

If AR devices were able to have a fuller semantic understanding of the user's environment, then the user's experience could be greatly improved. That is, in an ideal scenario, an AR device would know where various objects are in a scene (e.g. chair, table, book etc.) and be able to draw bounding boxes around them or even segment them in 3D space. Such a granular and fuller semantic understanding of the environment has the potential to be applied to:

- Visual output security for AR devices [2, 17, 37, 38, 55]. Knowing what objects are in a scene is important for effective visual output security and can also enhance the user experience in general by, for example, not occluding other users' faces when wearing a head mounted AR device.
- Better meshing. The meshes generated by current AR devices do not mesh surfaces very well (e.g. the meshes for flat tables tend to have unevenness). However, if the meshing algorithm knew that a certain region was a table, for example, it could smooth out the mesh. In this manner, semantic information could inform meshing algorithms to improve their meshes.
- Building a detailed semantic spatial map of the world (i.e. a "reality map") to improve the AR experience by seamlessly mixing the physical and virtual worlds [14]. For example, a reality map with accurate localization would allow an AR application to persist virtual objects in physical space.

The computer vision community, empowered by advances in deep learning thanks to bigger data sets, better algorithms and powerful compute, have, in recent years, made tremendous progress in image classification [23, 27, 35, 57, 65, 70], semantic segmentation [6, 56, 78, 80, 81] and object detection [20, 21, 41, 53, 54]. Consequently, the aim of this project is to investigate the application of such computer vision algorithms within the constraints of an AR deployment to provide an understanding of the environment for AR applications to exploit. This project also explores two applications which benefit from semantic understanding in AR: visual output security and semantically assisted meshing. The contributions of this project are:

- Training an indoor scene semantic segmentation model and deploying it on a physical AR device (Magic Leap One), providing a platform on which to build rich AR experiences that utilize the semantic information of a user's environment.
- Exploring the use of reinforcement learning (RL) for visual output security in AR by training a visual output security policy using RL and deploying it on a physical AR device (Magic Leap One). This work was published as an abstract and accompanying demo at ACM SenSys '19 [17].
- Proposing the use of semantic context to aid in mesh generation for AR devices.

In addition, the work presented here is being prepared for submission to the Fifth ACM/IEEE Symposium on Edge Computing as a position paper.

Section 2 will present work to bring semantic understanding to AR using a semantic segmentation model, edge server, and Magic Leap One. This work provides the foundation on which AR applications that utilize semantic information can be built. Section 3 then demonstrates visual output security, an application of semantic understanding in AR. Specifically, RL is used to train a visual output security policy which is deployed on the Magic Leap One. Section 4 offers a proposal for another application of semantic understanding in an AR context, semantically aware meshing, which uses semantic cues to improve mesh quality for AR use cases. Subsequently, section 5 discusses future research directions, and section 6 concludes the work presented here.

## 2 SEMANTIC UNDERSTANDING FOR AUGMENTED REALITY

### 2.1 Overview

The motivation to build a pipeline for providing semantic understanding to AR applications is fueled by the belief that a truly immersive AR experience requires a semantic understanding of the user's environment. However, providing such an understanding of the user's environment is challenging and requires the fusion of multiple algorithms and systems across disciplines. For example, building a detailed semantic map of the world would require simultaneous localization and mapping (SLAM) to automatically generate a map and a fusion of computer vision algorithms (e.g. object detection, semantic segmentation, and scene understanding) to overlay the map with semantic information. Further, due to the computational limitations of mobile devices, deploying high quality vision models with real-time performance will most likely require the use of edge computing.

While a semantic understanding of the world would be beneficial to building an immersive AR application, there is currently no platform which provides this kind of semantic information. Consequently, this project is aimed at building a proof-of-concept pipeline to provide semantic cues to AR applications to aid the research and development of AR applications that require semantic information. To this end, this investigation trains a 2D semantic segmentation model for indoor scenes and builds a system which projects the semantic information onto 3D space, communicates this information to a physical AR device (i.e. Magic Leap One), and overlays semantic information

in the user's field of view. All code used in this project is publicly available [16] as well as the preprocessed data and checkpointed model which had the best performance [15].

## 2.2 Related Work

*2.2.1 Image Classification.* AlexNet [35] is often referred to as the image classification network which ushered in the current era of convolutional deep learning methods applied to image classification tasks with very impressive results. VGG [65] studied the effects of deep convolutional networks for image classification and proposed the use of 16-to-19-layer deep networks, such as the VGG16 variant which has 13 convolutional layers and 3 dense layers.

While deeper networks tend to have higher image classification accuracy, the authors of ResNet [23] point out that deeper networks can be difficult to train. They proposed the use of residual layers and were able to train an image classification model with 152 layers. Even though complex models provide high classification accuracy, it is desirable to achieve high accuracy with more lightweight networks for mobile applications. MobileNet [27] and MobileNetV2 [57] both tackle this problem and introduce lighter weight image classification models which can also be used as a relatively lightweight backbone for object detection and semantic segmentation tasks. EfficientNet [70] also aims to provide more lightweight image classification models and their methodology produces state-of-the-art performance with fewer parameters than its counterparts.

*2.2.2 2D Object Detection.* Object detection is a difficult task because one must predict the class of an object as well as its location within an image for an arbitrary number of objects. While a naive approach may break the image into many regions and run classification on each region, such an approach is not ideal. R-CNN [21] introduced the notion of region proposals to combat this issue. In R-CNN, 2000 candidate region proposals are generated using selective search [73] and fed into a convolutional network to generate a feature map which is further fed into an SVM for final classification. Unfortunately, R-CNN does not have real-time performance, which led to the development of Fast R-CNN [20] and Faster R-CNN [54] which sped up the network using various optimizations.

Unlike R-CNN and its variants which rely on regions and do not examine the entire image, YOLO [53] splits an image into an $N \times N$ grid and classifies each grid square as belonging to a specific object. The Bounding boxes are generated along with these class probability which results in much faster performance. SSD [41] is another widely used object detection network and utilizes an image classification backbone (VGG16 in the original paper) and has high accuracy while maintaining good inference speed. SSD utilizes a set of default bounding boxes for which class probabilities are generated. These default bounding boxes go through a non-maximum suppression step which generates the final bounding boxes.

*2.2.3 3D Object Detection.* 3D object detection aims to generate bounding boxes for objects in 3D space. While some methods such as [40] and [12] use only a 2D input image to perform monocular 3D object detection without the need for RGB-D sensors, other methods employ the prevalence of RGB-D sensors and data to perform 3D object detection on point cloud data.

Frustum PointNets [51] utilize mature 2D object detection by first using 2D object detection to generate region proposals. These proposals are extruded to 3D space to generate frustums which are passed through PointNets [11] for further processing. Unlike Frustum PointNets, PointRCNN [62] is a model which performs 3D object detection from raw point clouds without the need of proposal generation from 2D object detection.

*2.2.4 2D Semantic Segmentation.* Semantic segmentation is the task of assigning each pixel a class label given an input image. The idea of using models composed of convolutions [42] has

significantly improved the accuracy of semantic segmentation models. SegNet [6] uses a VGG16 [65] image classification model as an encoder network and builds a decoder network which takes advantage of the features learned by the image classification model to output pixel-wise class predictions. While SegNet was originally benchmarked against outdoor driving scenes and indoor scenes, U-Net [56] is another semantic segmentation network which was originally proposed for biomedical imaging purposes, although it can be applied to other semantic segmentation tasks. U-Net builds upon the idea of fully convolutional networks [42] and has a similar encoder-decoder structure to SegNet. One key difference between the two networks is how encoded features are fed into the decoder network.

The authors of ICNet [80] note that although there has been much work to improve semantic segmentation model quality, there has not been as much focus on building models which provide high quality results with low latency. Specifically, their results illustrate that running ResNet38 [78] and PSPNet [81] models can take up to one second on a Nvidia TitanX GPU for a $1024 \times 2048$ image. Such a high latency for inference precludes the use of these models in a practical real-time setting. Their proposed model provides real-time performance while maintaining high quality results.

*2.2.5 3D Semantic Segmentation and Scene Understanding.* 3D semantic segmentation extends 2D semantic segmentation to provide semantic segmentation for point clouds. A challenge here is how to apply convolutions to sparse point cloud data to generate semantic labels using deep learning techniques. PointConv [77], PointNet [11] and PointNet++ [52] all provide solutions for this problem. Note that while these networks can be used for point cloud semantic segmentation, the algorithms they propose can also be applied to other problems such as 3D object detection.

Work related to 3D semantic segmentation is 3D scene understanding. Some methods utilize voxels and conditional random fields to generate semantic and geometric information about scenes [33] while more recent approaches such as JSIS3D utilize neural networks to perform both semantic segmentation and instance segmentation [50]. Further, as model performance on 3D segmentation tasks using point clouds have improved, recent work has moved toward more challenging problems such as fine-grained 3D semantic segmentation [46].

*2.2.6 SLAM.* Work in SLAM is critical to realizing a full semantic understanding of the environment. One can view an ideal semantic map of the world for an AR setting as a more ambitious version of the high definition (HD) maps [24, 72] being build for autonomous driving applications. For example, work to detect changes in HD maps [48] could be applied to building more detailed semantic maps for AR. In terms of semantic mapping in a SLAM setting, SLAM can aid in semantic segmentation by splitting the map generated by SLAM into semantic classes. Semantic information can also aid in SLAM by providing prior information about an object's geometry. In addition, joint SLAM and semantic inference can be performed to achieve both semantic segmentation and mapping using a joint algorithm [8]. In particular, work by Vineet et al. allows for real-time semantic outdoor scene reconstruction [76].

*2.2.7 Edge Computing.* Although the cloud has led to the development of many new services, the limitations of the cloud have become increasingly apparent as we have tried to develop new applications. Specifically, bandwidth and latency issues are particularly difficult to overcome in the world of mobile-cloud communication. Such constraints have fueled a recent re-focusing on edge computing technologies. Research has shown that cloudlets [58] (i.e. compute servers located physically closer to the end user, also referred to as edge servers) can aid in real-time video denaturing [68], cognitive assistance applications [13], and virtual as well as augmented reality [26]. The idea of bringing powerful compute closer to mobile devices to combat bandwidth and

latency issues has the potential to aid in many applications which require high compute while maintaining real-time performance.

### 2.3 System Architecture

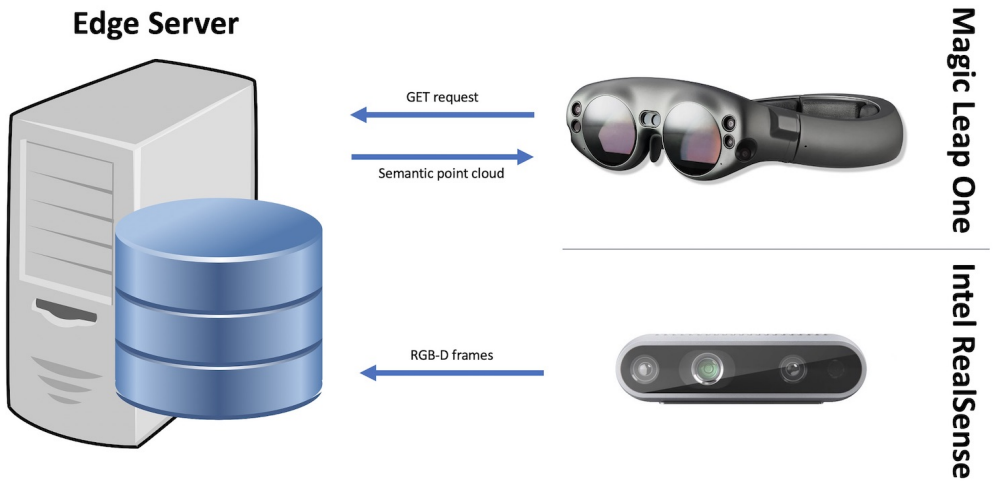


Fig. 1. System architecture for providing semantic information using a RGB-D camera and edge server in an AR setting. When the Magic Leap One makes a wireless GET request for semantic information, the edge server captures an RGB-D frame from the Intel RealSense using a wired connection and executes the semantic segmentation model. The edge server then projects the segmentation result onto the 3D point cloud and sends this semantic point cloud to the Magic Leap One wirelessly in response to the GET request.

The overall system architecture is presented in figure 1. The Magic Leap One [36] head mounted AR device is chosen as the physical AR device to use for deployment of the system and the Intel RealSense [28] RGB-D camera is used to collect RGB-D data. The segmentation model performs semantic segmentation on RGB data and projects it onto the point cloud. An edge server (Dell XPS desktop computer with Intel i7-9700 CPU, 16GB of RAM, and GeForce RTX 2060 GPU with 6GB of on-board memory) is used to execute the semantic segmentation model and project the result onto 3D space. This process is akin to how RGB point clouds are generated except that each point is also semantically classified. The AR device (i.e. Magic Leap One) can request segmented point cloud data from the server, and the server returns the annotated point cloud for rendering. Training of the semantic segmentation model is conducted on the edge server, although a real-world deployment would most likely train more powerful models in cloud data centers.

As figure 1 indicates, the Intel RealSense is connected to the edge server via a physical wire. Although ideally the Intel RealSense would be connected to the Magic Leap One, with the Magic Leap One transmitting RGB-D frames to the edge server for inference, due to limitations with the Magic Leap One this is currently not possible. In addition, although the Magic Leap One has a depth sensor as well as camera, information from these sensors cannot be used due to limitations in the current Magic Leap API. To request semantic information, the Magic Leap One makes a wireless asynchronous GET request to the server. Then, the edge server captures a RGB-D frame from the Intel RealSense through a wired connection and executes the semantic segmentation model and projects the result onto the 3D point cloud. The edge server transmits the result wirelessly to the

| | Red | Green | Blue |
|---|---|---|---|
| Mean | 0.491024 | 0.455375 | 0.427466 |
| Standard Deviation | 0.262995 | 0.267877 | 0.270293 |

Table 1. Channel-level mean and standard deviation of dataset.

Magic Leap One in response to the original GET request which renders the semantic point cloud in the user's field of view. A local wireless network using a NETGEAR Nighthawk X10 router is used for communication between the Magic Leap One and Intel RealSense.

### 2.4 Dataset

The SUN RGB-D [30, 63, 67, 79] dataset of indoor scenes was used to train the semantic segmentation model. The dataset contains 10335 RGB-D images. Only the RGB component was used to train the model. In addition to an "unknown" class, seven object classes were chosen from the dataset to train on: bookshelf, desk/table/counter (although these objects are distinct in the dataset, they were treated as one object class in this work), chair, book/paper (even though these objects are distinct in the dataset, they were treated as one object class in this work), picture, window, and door.

The dataset needed to be preprocessed and cleaned before it could be used. Quite a few of the JSON annotation files were corrupted due to misformatting and needed to be manually corrected. In addition, invalid annotations (e.g. annotations where the $x$ or $y$ coordinate was an invalid location) needed to be discarded of which there were 882. Further, class names were not consistent and contained spelling errors which required manual inspection to assign them to the correct class.

Images were center cropped to $224 \times 224$. Once cropped, 1226 images no longer contained any annotations because the cropped region did not have any annotated classes. The images with no annotations were discarded resulting in 9109 images which could be used for training and testing.

The images in the dataset had mean and standard deviation in table 1. Images were first normalized by applying the formula

$$\text{normalized image} = \frac{\text{original image} - \text{mean}}{\text{standard deviation}}$$

to each pixel's R, G, and B channel before feeding it into the semantic segmentation network for both training and inference. The annotations were preprocessed from polygon representations to per-pixel annotations. Hence, each annotation file contains a $224 \times 224$ array indexed $[h][w]$ where each element is the class ID associated with the $224 \times 224$ image. These annotations were turned into a one-hot encoding on the fly for training, since storing them as one-hot vectors would have used significantly more disk space.

The eight classes and their distribution in the dataset are shown in table 2. The distribution for a given class is the proportion of pixels a specific class appears in across the entire dataset. It is important to note the significant imbalance in the dataset as exemplified by table 2. This imbalance was accounted for during training and the methods used are described in the Model Training section. The data (9109 images) were randomly split into a training set (table 3) and a test set (table 4) using a random 90/10 split resulting in 8198 training images and 911 test images (note that both sets of data have similar class distributions to each other and the overall dataset, as desired).

It is also important to point out that the SUN RGB-D dataset has incomplete annotations. Take the example in figure 2. In this example, the annotation for the pictures on the wall are missing (see legend in figure 11). Missing annotations can result in a model which does not semantically segment objects as well as expected in the real-world because the model is evaluated on data

| Class | ID | # of Instances | % of Pixels |
|---|---|---|---|
| Unknown | 0 | 336035069 | 73.52% |
| Bookshelf | 1 | 2273833 | 0.50% |
| Desk/Table/Counter | 2 | 47156248 | 10.32% |
| Chair | 3 | 44957965 | 9.84% |
| Book/Paper | 4 | 2891720 | 0.63% |
| Picture | 5 | 2838463 | 0.62% |
| Window | 6 | 11808008 | 2.58% |
| Door | 7 | 9091878 | 1.99% |

Table 2. Per class distribution for the dataset. The distribution is the proportion of pixels for a given class across the entire dataset.

| Class | ID | # of Instances | % of Pixels |
|---|---|---|---|
| Unknown | 0 | 302072880 | 73.44% |
| Bookshelf | 1 | 2091497 | 0.51% |
| Desk/Table/Counter | 2 | 42791040 | 10.40% |
| Chair | 3 | 40423443 | 9.83% |
| Book/Paper | 4 | 2696598 | 0.66% |
| Picture | 5 | 2538673 | 0.62% |
| Window | 6 | 10658005 | 2.59% |
| Door | 7 | 8070712 | 1.96% |

Table 3. Training set (8198 randomly chosen images) per class distribution. The distribution is the proportion of pixels for a given class across the training set.

| Class | ID | # of Instances | % of Pixels |
|---|---|---|---|
| Unknown | 0 | 33962189 | 73.97% |
| Bookshelf | 1 | 182336 | 0.40% |
| Desk/Table/Counter | 2 | 4365208 | 9.51% |
| Chair | 3 | 4534522 | 9.88% |
| Book/Paper | 4 | 195122 | 0.42% |
| Picture | 5 | 299790 | 0.65% |
| Window | 6 | 1150003 | 2.50% |
| Door | 7 | 1021166 | 2.22% |

Table 4. Test set (911 randomly chosen images) per class distribution. The distribution is the proportion of pixels for a given class across the test set.

that contains errors. Further, even if the model learned to segment certain objects well, the final evaluation metrics may not reflect this because the test data is missing some annotations.

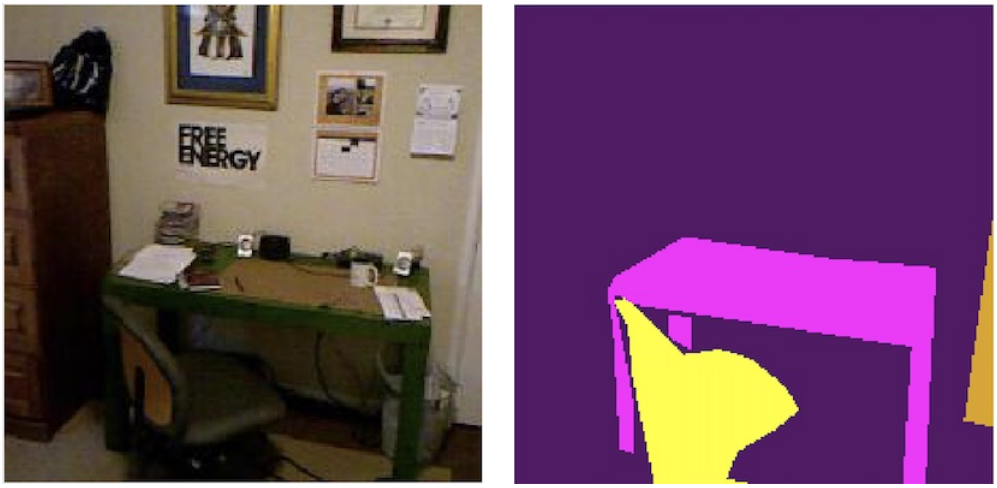


Fig. 2. Example of missing annotations from the SUN RGB-D dataset. Note that the annotation of the pictures are missing on the right (see legend in figure 11). Missing annotations can result in a lower quality model when training. In addition, the accuracy and other quality metrics of the model may not be completely indicative of real-world performance since the underlying data used to evaluate the model contains errors.

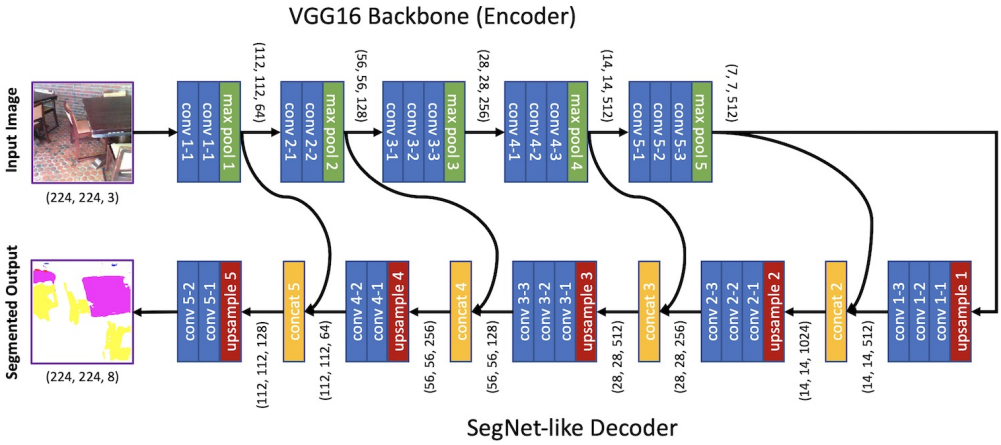


Fig. 3. The semantic segmentation model which was used. The blue “conv" layers are a convolution + batch normalization + ReLU activation except for the last conv 5-2 layer in the decoder network which uses a softmax activation. The red layers are upsample layers and yellow “concat" layers are a concatenation of the two inputs which stack one input on top of another.

## 2.5 Model Architecture

The architecture of the semantic segmentation network is illustrated in figure 3. The input to the model is a $224 \times 224 \times 3$ image and the output is a $224 \times 224 \times 8$ matrix where there is an eight element vector for each pixel. The index of the eight element vector with the highest value (i.e. probability) corresponds to the predicted class of the pixel. Because batching is supported, the actual input size of the model is $N \times 224 \times 224 \times 3$ and the actual output size is $N \times 224 \times 224 \times 8$ for a given batch size $N$.

The model architecture used is similar to SegNet [6]. A pre-trained VGG16 image classification network, with the final three dense layers removed, is used as the encoder backbone of the model. In addition, the decoder network mirrors the VGG16 backbone just like in SegNet. The key difference is that instead of storing the max pooling indices and using them to upsample decoder network feature maps to produce sparse feature maps, the network used in this work simply concatenates the encoder output with the corresponding input in the decoder network. This kind of concatenation has similarities with U-Net's architecture [56].

Note that the input image size is constrained by the underlying encoder network (i.e. the pre-trained VGG16 model) and that changing it requires swapping out the encoder network with a model which can take in other input image sizes. However, the number of output channels (i.e. number of classes) can be set to any number. Further, the underlying encoder network can theoretically be switched out for other models such as ResNet [23].

### 2.6 Model Training

Training was conducted on a GeForce RTX 2060 GPU with 6GB of on-board memory for 60 epochs. Total training, at worst, took 12 hours for a batch size of 2. However, training was faster when smaller batch sizes were used. Batch size was constrained by the GPU memory. Specifically, collecting additional metrics during training required more GPU memory which limited the batch size.

Training the semantic segmentation model was complicated by the significant data imbalance. To account for the data imbalance, weighted categorical cross entropy was used as the loss function. This loss was calculated using the function

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C}\alpha_c 1_{y_i \in C_c}\log P_{\text{model}}(y_i \in C_c) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C}\alpha_c 1_{y_i \in C_c}\log \hat{y}_i$$

where $N$ is the number of pixels in a batch, $C$ is the number of categories, $y_i$ is the ground truth for the i-th pixel, $\hat{y}_i$ is the prediction for the i-th pixel, $C_c$ is the category with ID $c$, and $\alpha_c$ is a weight associated with class $c$. The weights $\alpha_c$were calculated using the median frequency method [18]. Specifically,

$$\alpha_c = \frac{median\_freq}{freq(c)}$$

where $freq(c)$ is the number of pixels with class $c$ divided by the total number of pixels in images where class $c$ is present. The $median\_freq$ is the median of the all the $freq(c)$ which are calculated for each class. The weights used for each class are listed table 5. As the table illustrates, the more prevalent classes have lower weights, which helps in balancing the loss function so that the model learns to classify rare objects as well as common objects.

Data was also augmented during training to improve the generalizability of the model. A horizontal flip was applied randomly with probability 0.5. An additional rotation or horizontal shear was applied with probability 0.5 each. For rotations, images were rotated by some random angle between $-20°$ and $20°$. For horizontal shears, a random shear between $-0.2$ and 0.2 was applied.

As figure 4 indicates, the model training was successful with both the accuracy increasing and the loss decreasing in a smooth manner. A detailed evaluation of the model is presented in the Model Evaluation section.

### 2.7 Deploying the System

*2.7.1 Edge Server Setup.* An edge server was developed using the Flask library for python to serve as a compute node to receive input from the Intel RealSense RGB-D camera and process the data. The edge server is configured to send the AR application a JSON object containing locations of 3D

| Class | ID | Weight |
|---|---|---|
| Unknown | 0 | 0.14385866807 |
| Bookshelf | 1 | 0.90557223126 |
| Desk/Table/Counter | 2 | 0.72059989577 |
| Chair | 3 | 0.66989873421 |
| Book/Paper | 4 | 3.53100851804 |
| Picture | 5 | 2.53715235032 |
| Window | 6 | 1.11641301038 |
| Door | 7 | 1.18609654356 |

Table 5. The class weights used in the loss function for training.

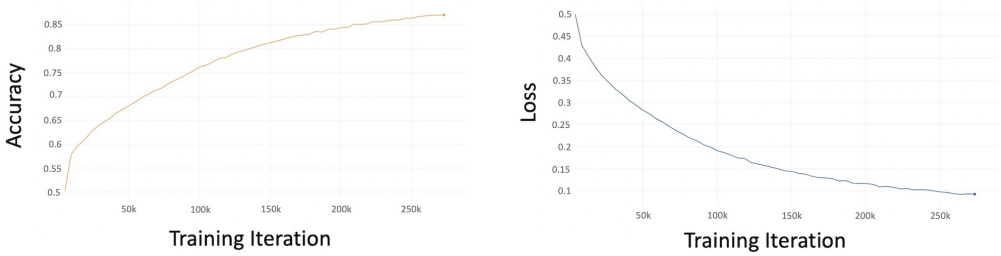


Fig. 4. Training accuracy and loss plotted against training iteration. A single training iteration constitutes one batch.

“pixels” with corresponding semantic information (i.e. a semantic point cloud). Only points which have a semantic label other than “Unknown" are sent to the Magic Leap One for rendering to save bandwidth since the Magic Leap One only renders the seven known semantic classes. The JSON file which is transmitted has the following format:

```
{
    "locations": [
        {
            "x": FLOAT,
            "y": FLOAT,
            "z": FLOAT,
            "pixelClass": CLASS_ID_INT
        },
        ...
        {
            "x": FLOAT,
            "y": FLOAT,
            "z": FLOAT,
            "pixelClass": CLASS_ID_INT
        }
    ]
}
```

The Magic Leap One can query the server for the semantic point cloud using an http GET request over the local network. Once the server receives a GET request it captures an RGB-D frame from the Intel RealSense, runs inference on the semantic segmentation model, and projects the result onto 3D space. The resulting semantic point cloud is sent back to the Magic Leap One.

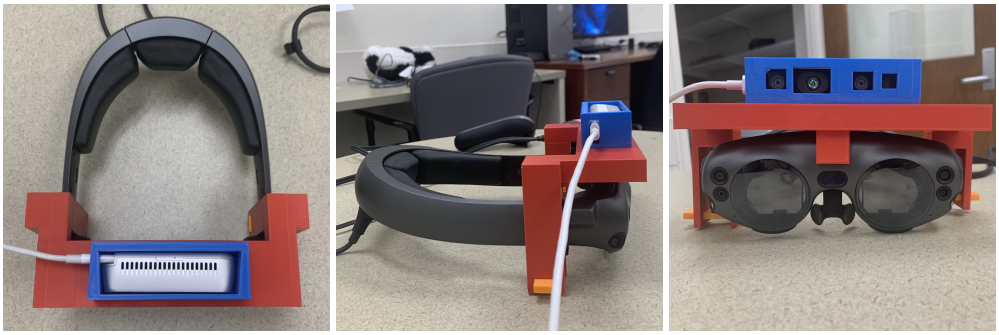

Fig. 5. Custom mount used to mount the Intel RealSense on the Magic Leap One.

*2.7.2 Custom Mount and Device Calibration.* A custom mount was designed using CAD software and printed using a 3D printer to mount the Intel RealSense camera on the Magic Leap One (see figure 5). The mount consists of a cradle for the Intel RealSense camera (blue component) which fits on top of the red support component that is mounted on the Magic Leap One. While the mount and camera increase the weight on the overall device, there is no other ergonomic change as a result of wearing the Magic Leap One with the custom mount attached.

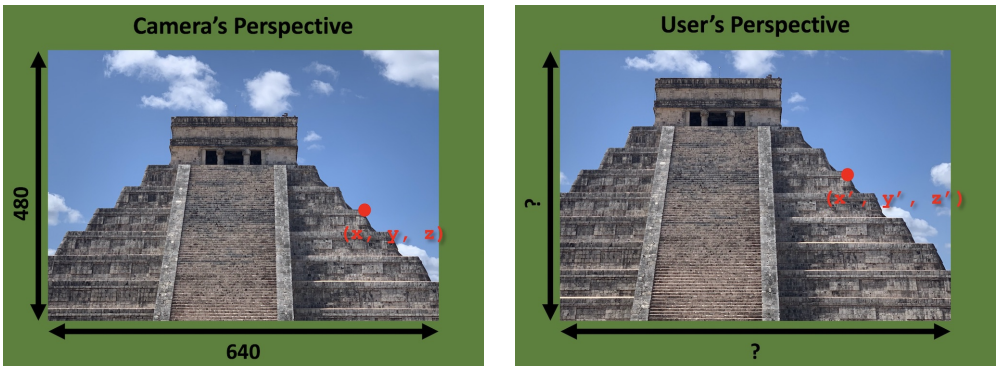


Fig. 6. The mismatch between what the Intel RealSense camera captures (left) and what the user sees (right).

Once the mount was designed and 3D printed, the Intel RealSense camera had to be calibrated to the Magic Leap One's field of view. The calibration problem is depicted in figure 6. The Intel RealSense camera captures a $640_x \times 480_y$ RGB image with depth values measured in meters. The semantic segmentation algorithm center crops the camera frame to a $224_x \times 224_y$ image and projects the semantic labels onto the depth data to generate a $224_x \times 224_y \times depth_z$ semantic point cloud.

Because of misalignment between what the camera captures and the user's field of view, the $(x, y)$ coordinates of the point cloud cannot be simply scaled to the Magic Leap's coordinate system. Since the Intel RealSense camera is mounted on top of the Magic Leap One, there is a vertical

misalignment between the camera's frame and user's field of view (i.e. the camera captures more information in the vertical direction above the user's field of view). In addition, because the Intel RealSense camera's RGB sensor is not centered exactly in the middle, there is also a horizontal misalignment between the camera's frame and user's field of view (i.e. the camera captures more information to the left of the user's field of view). Therefore, the calibration process needed to apply $x$ and $y$ transforms to the semantic point cloud to align it with the user's field of view. In addition to transforming the $(x, y)$ coordinates the $z$ value also needed to be transformed because the Magic Leap uses a different coordinate system. The transform values were obtained through manual inspection. The transformation for a $(x, y, z)$ semantic point cloud point to the user's field of view is,

$$z' = \frac{z - 0.37}{1.5},$$
$$x' = x - 100,$$

and

$$y' = y - 100.$$

While the transformed $z'$ can be used directly in the Magic Leap's coordinate system, the $x$ and $y$ values need to be further normalized to the Magic Leap's coordinate system (i.e. while the width and height of the semantic point cloud is 224 pixels, the Magic Leap's view's width and height are not, as indicated by the question marks in figure 6). Obtaining this normalization factor proved more difficult than expected because, due to the nature of the Magic Leap's API and its integration with Unity, the normalization factor depends on the depth $z$.

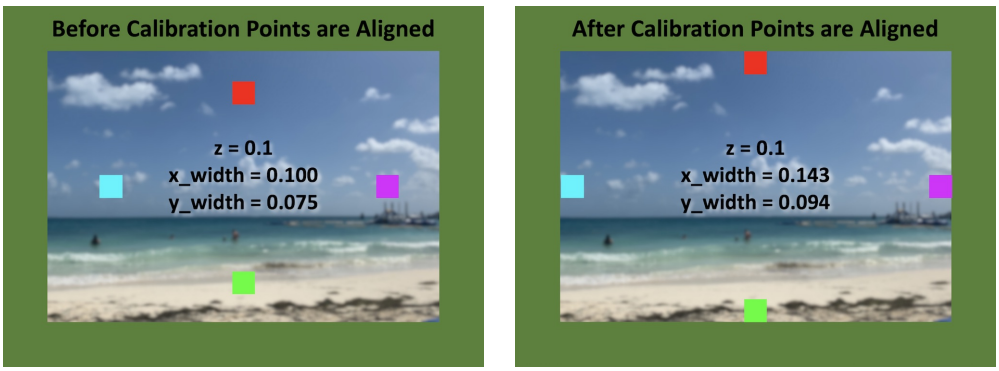


Fig. 7. An illustration of the calibration application (this not a screenshot of the application).

A custom Magic Leap application was developed to derive a function which maps $z$ values to the $x$ and $y$ normalization factors. Figure 7 illustrates this calibration application. The user can use a hand-held controller to change the $x\_width$ and $y\_width$ of a given $z$ value until the four colored alignment squares align with the outer edge of the user's field of view. Using this calibration application, the $x\_width$ and $y\_width$ values were obtained for $z$ values ranging from 0 to 1 in 0.1 increments. The data points were plotted and best-fit line calculated to produce a function to calculate $x\_width$ and $y\_width$ given an arbitrary $z$ value (see figure 8). Note the linear relationship between $z$ values and $x\_width$ and $y\_width$. From the data, the following functions were derived based on the linear line of best fit:

$$x\_width(z) = 0.725 \times z + 0.0649$$

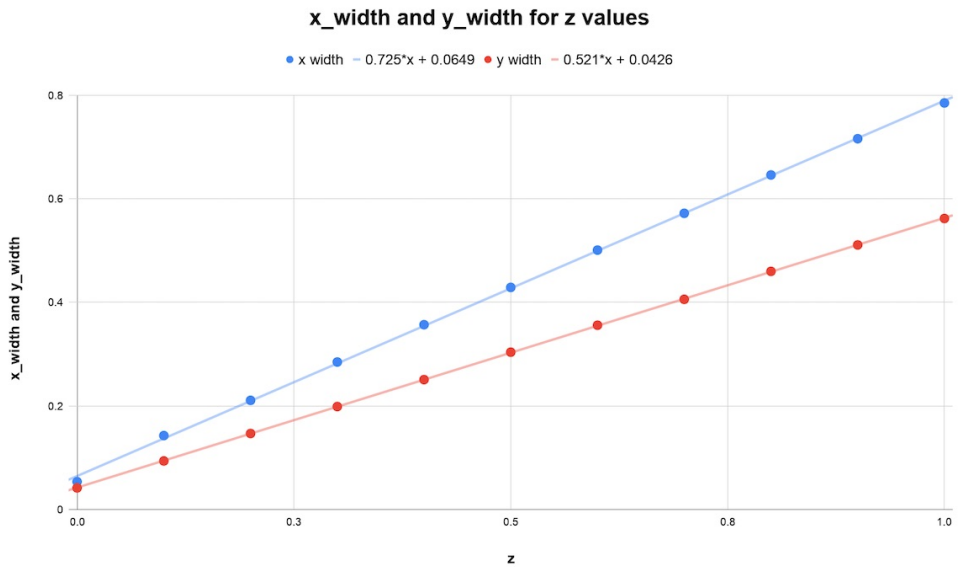


Fig. 8. Result of manually calibrating the normalization factors $x\_width$ and $y\_width$ with a custom calibration application.

and

$$y\_width(z) = 0.521 \times z + 0.0426.$$

Using the above functions, the final transformation for the $(x, y)$ coordinate of the semantic point cloud to the $(x, y)$ coordinate of the Magic Leap given a transformed $z$ value $z'$ is

$$x' = \frac{x - 100}{x\_width(z')}$$

and

$$y' = \frac{y - 100}{y\_width(z')}.$$

*2.7.3 Overlaying Semantic Information in the User's Field of View.* An AR application was developed for the Magic Leap One which can query the edge server for the semantic point cloud and display it in the user's field of view. Examples of what the user sees are in figure 9. Note that the semantic overlay is slightly misaligned in the screenshots because the Magic Leap One's screenshot function does not correctly align holograms to match what the user sees.

Displaying the semantic overlay requires rendering thousands of points efficiently. The Unity Mesh object was used to accomplish this because the Magic Leap One is optimized to display Unity Mesh objects. Normally, a mesh is composed of vertices which define surfaces to be rendered. However, in this case a custom shader was used to render just the vertex points efficiently without rendering surfaces.

## 2.8 Model Evaluation

Figure 10 contains ten examples of outputs from the trained semantic segmentation model along with the corresponding input image, normalized image, and true mask. Per class accuracy, balanced accuracy and IoU are presented in table 6.

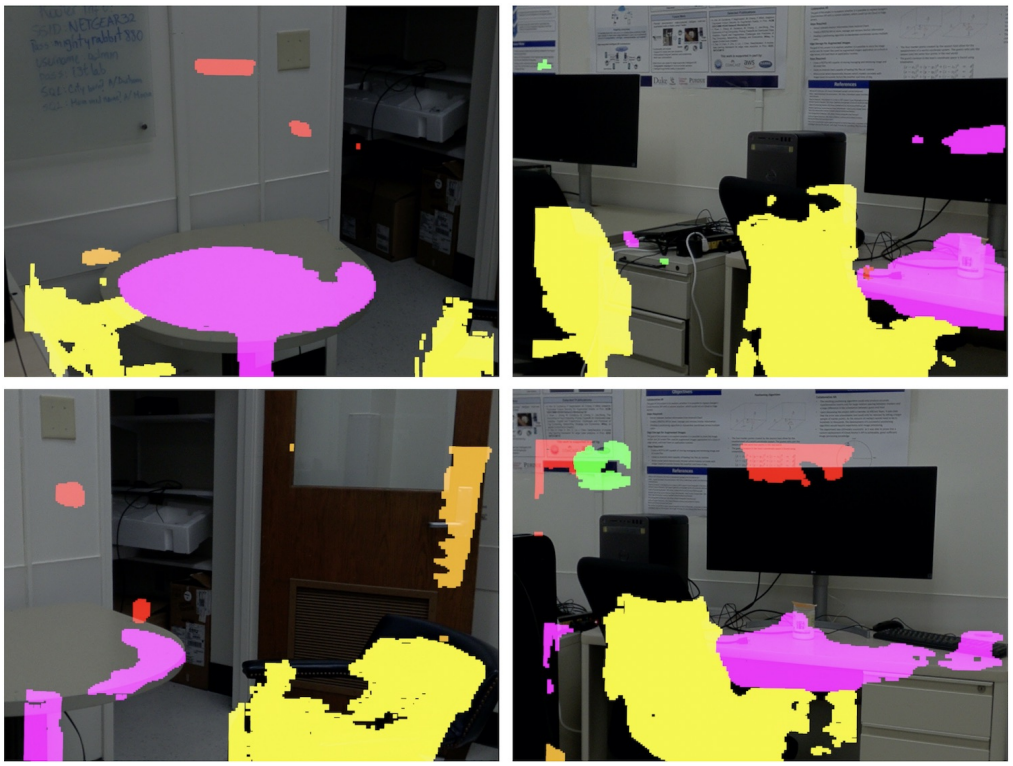

Fig. 9. Screenshots of the Magic Leap One application which was developed to overlay semantic information in the user's field of view. Note the misalignment of semantic information is an artifact of the Magic Leap One's limited screenshot capability. The semantic overlay is much better aligned when viewed using the Magic Leap One.

| Class | ID | Accuracy | Balanced Accuracy | IoU |
|---|---|---|---|---|
| Unknown | 0 | 26.44% | 50.33% | 0.1357 |
| Bookshelf | 1 | 99.55% | 49.98% | 0.5006 |
| Desk/Table/Counter | 2 | 82.88% | 60.31% | 0.4934 |
| Chair | 3 | 78.22% | 70.37% | 0.4901 |
| Book/Paper | 4 | 99.57% | 50.00% | 0.4986 |
| Picture | 5 | 99.34% | 50.01% | 0.5054 |
| Window | 6 | 40.86% | 65.88% | 0.2203 |
| Door | 7 | 97.74% | 50.08% | 0.4911 |

Table 6. Per class accuracy, balanced accuracy, and IoU of the semantic segmentation model on the test set.

Overall the model learned quite well, as can be seen qualitatively from figure 10. Although the model has some weaknesses in outputting clear edges when segmenting objects (see the bottom left example in figure 10 where the group of chairs is not distinguishable in the model output), the model nonetheless segments different objects quite well. For example, the ground truth label in the second image from the top of the left column in figure 10 is missing annotations for the pictures on

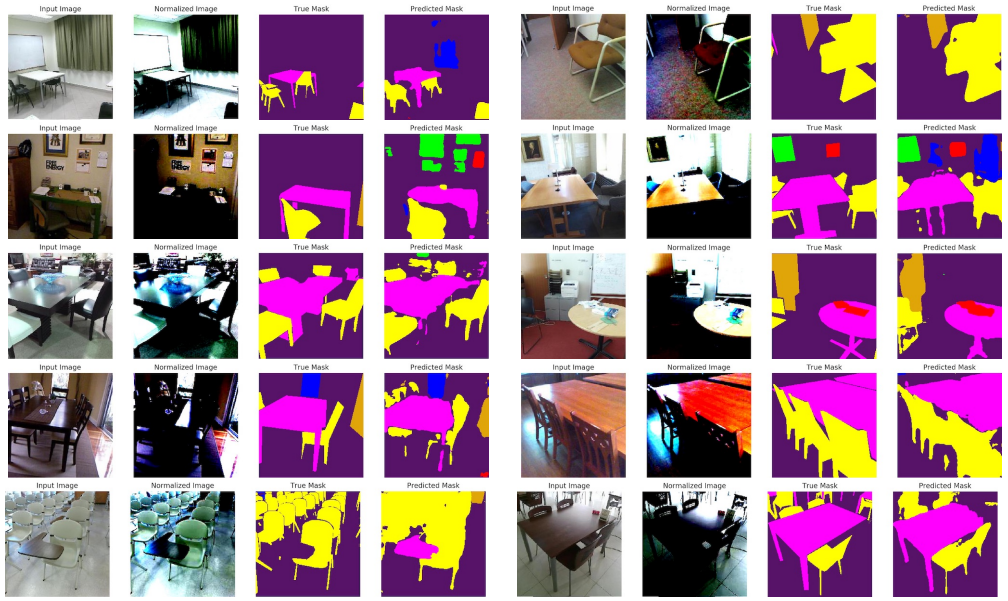


Fig. 10. Examples of input image, normalized input image, true mask and predicted mask obtained from the trained semantic segmentation model. See legend in figure 11.

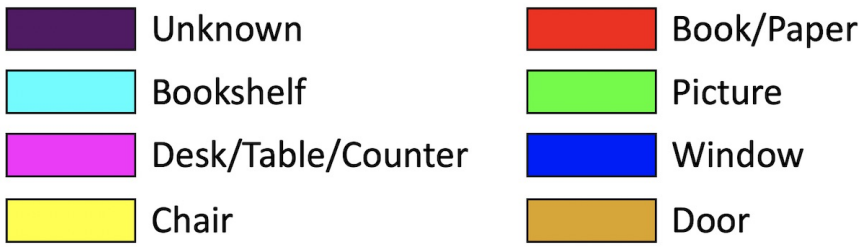


Fig. 11. Legend for segmentation class colors. When semantic information is displayed on the Magic Leap One the “Unknown" class is not displayed.

the wall. However, the model predicts their presence and segments them with reasonable quality. In another such example, the ground truth label in the second image from the top of the right column in figure 10 is missing annotations for the window but the model segments the window quite well. Further, the ground truth label in the bottom image of the left column in figure 10 does not contain an annotation for the table attached to the chair in the foreground but the model clearly segments the table. This qualitative analysis shows that the semantic segmentation model was able to learn even though the dataset it was trained on contains missing annotations. Moreover, the model is able to segment objects correctly even for objects where the ground truth label is missing the annotation.

The quantitative data from table 6 must be viewed in the context that the test set ground truth labels contain missing annotations due to the imperfections in the dataset. Therefore, even though some metrics may seem low, it is possible that the model is actually predicting objects well, as

observed qualitatively above in cases where the model correctly segments objects for which the annotation is missing in the ground truth label.

The raw per-pixel accuracy numbers from table 6 are very high for some classes (i.e. bookshelf, book/paper, picture, door) most likely because the accuracy is skewed towards correct classification of true negatives. The balanced accuracy presents a more complete picture in which bookshelf, book/paper, picture and door all have lower accuracy than desk/table/counter, window, and chair. The IoU is consistent among the classes and is similar to the overall IoU reported by the authors of SegNet [6]. However, the IoU for the unknown class and windows is significantly lower than the other classes. The model segmenting objects which are not annotated in the ground truth label may be the cause the low IoU for the unknown class. Nonetheless, the relatively high IoU metric for most classes and high balanced accuracy for chairs and desk/table/counter illustrate that the model segments indoor objects fairly well. Further, the data from table 6 also suggest that the model generalized to new input with the aid of the data augmentation discussed above.

### 2.9 System Performance

The round trip latency from the Magic Leap One requesting semantic information and to receiving it hovered at around 500ms. Unfortunately, this did not allow for real-time performance. The high latency was most likely due to two factors: an unoptimized network and the model being executed on the CPU. There was high round trip network latency (in the 100s of ms) even when there was no payload and no inference being performed, and the observed latency varied from as low as 50ms to as high as 1000ms.

Running model inference on the GPU would likely significantly improve the inference latency, bringing down the overall round trip time. In addition, network optimizations may help in bringing down overhead, reducing the overall system latency as well. Nonetheless, the semantic overlay, once calibrated, was of quite high quality and even the depth of the overly could be perceived because of its depth component.

### 2.10 Discussion

The SegNet [6] and U-Net [56] inspired semantic segmentation model with a VGG16 [65] backbone which was trained produces qualitatively good semantic segmentations of indoor scenes. While some object classes had lower accuracy, the model also learned to segment objects better than the ground truth label in some cases, demonstrating its robustness. The model's performance on the test set also illustrates its generalizability to indoor scenes which were not in the original dataset, and semantic segmentation output for the model worked well in the novel lab environment, even though images from the lab were not present in the dataset (see figure 9 for examples of the model semantically segmenting the lab). Training was successful even in the presence of large class imbalance of the data which was accomplished by using weighted categorical cross entropy as the loss function with weights calculated using the median frequency method [18].

A custom mount was built from scratch and 3D printed to mount the Intel RealSense camera on the Magic Leap One. Once a suitable mount was designed and printed, the Intel RealSense's RGB-D frames had to be calibrated with the Magic Leap One in order to overlay semantic information in front of the user's field of view. This calibration utilized a custom-built application to map the Intel RealSense's coordinate system to the Magic Leap's. The discrepancy in field of view between the Intel RealSense's sensors and the user had to be accounted for as well.

The edge server was built using python and received requests from the Magic Leap One for semantic information. Once a request was received, the server captured an RGB-D frame from the Intel RealSense, ran inference using the semantic segmentation model, and projected the result onto the 3D space to produce a semantic point cloud. The semantic point cloud was then sent

back to the Magic Leap One. Even though the round trip time of the Magic Leap One making a request to receiving a response did not enable a high frame rate real-time experience, it is likely that optimizing the network and model inference (e.g. running inference on the GPU instead of the CPU) will significantly improve the system's ability to execute in real-time. Furthermore, using the Magic Leap One's optimized mesh renderer with a custom shader to render vertex points instead of surfaces ensured that the semantic point cloud could be rendered without noticeable lag in the user's field of view to overlay semantic information.

Overall, this work achieved the goal of building a pipeline to bring semantic understanding to AR. The system provides a physical AR device (i.e. Magic Leap One) which a user can wear comfortably that has access to a semantic point cloud with the aid of an external sensor (i.e. Intel RealSense) and edge server. Such a system can be used to investigate applications in AR which require semantic understanding. Two such applications will be explored in the following sections.

## 3 ADAPTIVE AUGMENTED REALITY VISUAL OUTPUT SECURITY USING REINFORCEMENT LEARNING

### 3.1 Overview

While the increasing proliferation of AR devices will undoubtedly enable many new applications, issues of privacy and security cannot be ignored. This section will present an application of semantic understanding for AR: visual output security. In particular, an understanding of the user's environment is needed to decide what real-world objects are import in order to provide visual output security. For example, if a stop sign is detected the AR device may apply the visual output security policy to not obscure the stop sign. The policy presented here is trained using reinforcement learning (RL).

Much of the previous work on AR privacy has focused on the inputs to AR devices (i.e. input security) [29, 55]. Noting this gap in AR security research, Lebeck et al. suggested in a position paper that there should also be a focus on output security to secure the output of AR devices [37].

In particular, visual AR output security is concerned with two issues pertaining to the user's visual field:

- Regulating visual content displayed to the user to reduce distraction and obstruction of the real-world context.
- Preventing holograms from obscuring other holograms with a higher priority.

For example, in the case of displaying holograms in car windshields, it would be dangerous for a hologram to obstruct a stop sign. Similarly, a hologram which displays the speedometer should not be obstructed by a hologram which displays the album art of the song the driver is currently playing. In the case of ensuring important real-world context is not obstructed by a hologram, the AR device must have a semantic understanding of the world to know what regions of the user's field of view should be obstruction free. Consequently, visual output security is an application which greatly benefits from, and to a large degree requires, semantic information of a user's environment.

Although previous work has exemplified the importance of visual output security and demonstrated its feasibility in simulation [2, 38], they have not deployed visual output security policies on a physical AR device to ascertain its viability in the real world. In particular, whether a RL based output security model can be deployed without degrading performance was left an open question [2]. To fill this gap, this work investigates whether RL models can be deployed on a physical AR device without a noticeable degradation in performance and tests the deployment of a visual output security policy trained using RL.

It is important to note that while this work focuses on visual output security, there are concerns that other AR output, such as audio and haptic output, may need to be regulated as well [2, 38]. In

addition, one may also want to limit distracting and uncomfortable AR output such as blinking holograms, much like web browsers have evolved to block popups and the blink tag [38]. Further, the issue of visual output security can be seen as something which generally improves the user's experience in addition to providing security. For example, a user may want their AR device to automatically detect faces and not obstruct them.

This work using reinforcement learning for visual output security and deploying a system on a physical AR device was published as an abstract and accompanying demo at ACM SenSys '19 [17].

## 3.2 Related Work

*3.2.1 Augmented Reality Output Security.* Previous work has argued that while one could leave output security concerns to the application developer, it is much safer to have the OS guarantee the security of AR device outputs and investigated what an OS level AR device output security module may look like [37, 38]. They allowed developers to write policies which were enforced by Arya, a simulated proof of concept for an OS-level visual output security module.

However, as noted by Ahn et al., these hand-coded policies, while promising, can be difficult to define for real-world use [2]. For example, specifying a policy to move holograms that are obstructing an important real-world object while not moving holograms too far from their original location and at the same time not obscuring other important holograms is not trivial. Ahn et al. proposed the use of RL to solve this problem by automatically generating policies, and they demonstrated their approach's effectiveness in simulation [2]. More recently, Ahn et al. expanded the policy generation for visual output security by using imitation learning [1].

*3.2.2 Deep Reinforcement Learning.* Reinforcement learning is a trial-and-error model training technique based on behaviorist psychology [69]. Deep reinforcement learning is a method for reinforcement learning which uses neural networks to train agent policies, and it faces three main challenges [5]:

- The only signal the agent receives during training is the reward.
- An agent's observation can contain strong temporal correlations.
- Agents have to be able to overcome long-range time dependencies.

Deep reinforcement learning has been successfully used to solve complex problems such as playing classic Atari 2600 games [45], 3D bipedal and quadrupedal locomotion [59], and playing Go [64]. While various deep reinforcement learning algorithms have been proposed [5], the proximal policy optimization algorithm (PPO) has been found to be easier to implement, more general, and have better sample complexity [60].

## 3.3 Reinforcement Learning Trained Visual Output Security Policy

An initial experiment was conducted using the Magic Leap One to ascertain whether a RL model can be deployed on a physical AR device without degrading the user experience. Since responsiveness is essential to a good AR experience, computation which may decrease the frame rate of the AR device, such as running a RL model, may be detrimental to the user experience. For the experiment, a simple game was developed using Unity [75] which an agent was trained to play using RL. Training was conducted using Unity's ml-agents framework [74].

Although there were some challenges with ml-agent and Magic Leap One SDK configurations to deploy the RL policy on the Magic Leap One, once deployed the RL agent successfully played the game without any noticeable performance degradation, laying the foundation to train and deploy a RL trained visual output security policy on a physical device.

The output security problem for this project was defined as having one important real-world object and one or more holograms. The position of the important real-world object and holograms

were randomly assigned and the important real-world object was always placed behind holograms, although the holograms did not necessarily obscure the important real-world object. The width and height of the holograms were also randomly assigned in some cases.

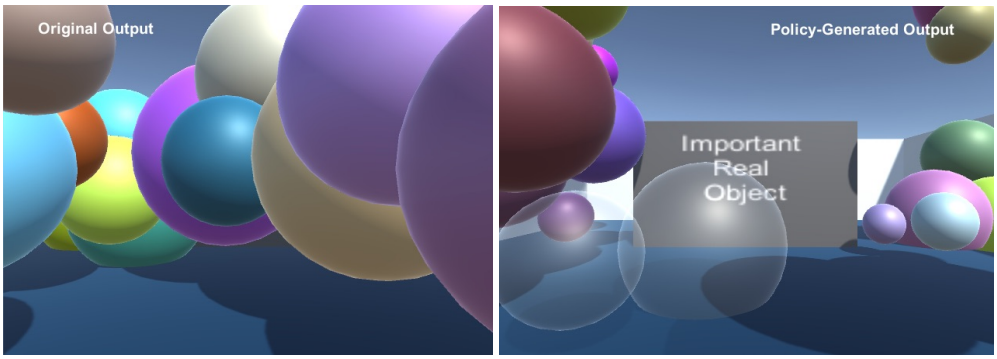


Fig. 12. Example of visual output before (left) and after (right) a RL generated visual output security policy is applied (from Ahn et al. [2]).

Given these conditions, the goal is to train an agent to move the holograms so that they do not obscure the important real-world object while keeping the holograms as close to their original positions as possible. Note that not obscuring other holograms was not taken into account. Figure 12 is an example of a simulated visual output security environment before and after RL-generated policies are applied.

*State Space.* A model where the agent has complete knowledge of the states was used. Two state spaces were defined:

- $S_1$ in which each $s \in S_1$ consists of the location ($x$ and $y$ coordinate), width, and height of the holograms as well as the important real-world object. That is, $S_1$ has $4(N+1)$ observations at each time step where $N$ is the number of holograms.
- $S_2$ in which each $s \in S_2$ consists of the location ($x$ and $y$ coordinate) of the important real-world object and the location ($x$ and $y$ coordinate), $x$-velocity, and $y$-velocity of the holograms. That is, $S_2$ has $4N+2$ observations at each time step where $N$ is the number of holograms.

*Action Space.* The RL agent was tasked with moving hologram(s) away from important real-world objects. Two action spaces were used:

- $A_1$ where the agent outputs the $x$ and $y$ coordinate for each hologram. That is, $A_1$ has $2N$ actions at each time step where $N$ is the number of holograms.
- $A_2$ where the agent outputs the $x$ and $y$ force for each hologram. That is, $A_1$ has $2N$ actions at each time step where $N$ is the number of holograms.

*Reward Function.* Initially, the following reward function (adapted from Ahn et al. [2]) with various values of $\alpha$ and $\beta$, where $r$ is the position of the important real-world object, $O$ is the set of original hologram positions, and $O'$ is the set of new hologram positions was used:

$$R_1 = \alpha \sum_{o' \in O'} Dist_{x,y}(o', r) - \beta \sum_{o \in O, o' \in O'} Dist_{x,y}(o, o').$$

This reward function is meant to move holograms away from the important real-world object while keeping holograms as close to their original position as possible. Although the suggested

values of $\alpha = 2.0$ and $\beta = 1.5$ [2] were tried, the reinforcement learning model did not converge in some cases, most likely because of hyperparameter issues.

The following four states were defined to make a simpler reward function which worked well:

- Success: The hologram does not obscure the important real-world object and has not moved outside of the user's field of view.
- Failed: The hologram has moved outside of the user's field of view.
- Incomplete: The hologram obscures the important real-world object and has not moved outside of the user's field of view.
- Done: The training session ended without reaching a Success or Failed state.

Two reward functions were defined using these three states.

*Reward Function $R_2$:*

$$R_2 = \begin{cases} 1, & \text{State = Success} \\ 0, & \text{State = Failed} \\ \text{continue session}, & \text{State = Incomplete} \\ 0, & \text{State = Done} \end{cases}$$

$R_2$ rewards the agent for moving a hologram away from a real-world object while ensuring it is visible to the user. In the case that there is more than one hologram, the reward was accumulated for each hologram and divided by the number of holograms to normalize the reward.

*Reward Function $R_3$:*

$$R_3 = \begin{cases} \frac{M-S}{M}, & \text{State = Success} \\ 0, & \text{State = Failed} \\ \text{continue session}, & \text{State = Incomplete} \\ 0, & \text{State = Done} \end{cases}$$

where $M$ is the max steps and $S$ is the current step. $R_3$ rewards the agent for moving a hologram away from a real-world object while ensuring it is visible to the user in a timely manner by giving less reward the more steps it takes. In the case that there is more than one hologram, the reward was accumulated for each hologram and divided by the number of holograms to normalize the reward. Furthermore, in this instance, the Success state requires that the hologram is in the same relative location to the important real-world object it was originally in. For example, if the hologram was originally at the top left of the important real-world object, it would have to move away from the important real-world object while remaining in the top left region relative to the important real-world object.

### 3.4 Visual Output Security Policy Training

Figure 13 provides an overview of the visual output security model training and deployment process. The model is trained using reinforcement learning in a simulated environment and deployed to the physical device (Magic Leap One) once training converges.

In the simulated environment, a green cube, which had a consistent scale, represented the important real-world object (see figure 14). It was placed randomly within the user's field of view at the beginning of each training cycle. Spherical holograms represented the objects which needed to be moved away from the important real-world object.

Two types of RL output security policies were trained. The first policy calculated where to place the holograms while the second calculated the direction in which to move the holograms.

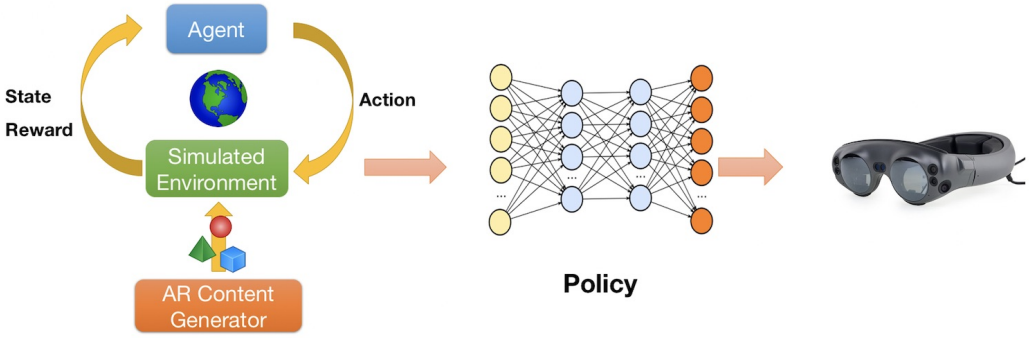


Fig. 13. General overview of the model training and deployment process (from Ahn et al. [2])

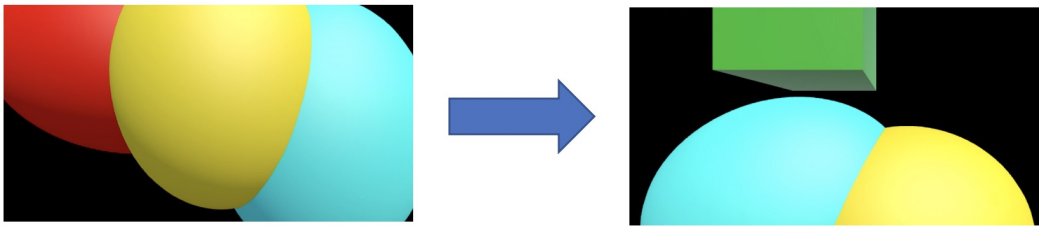

Fig. 14. Example of visual output before and after model-generated direction policy is applied to three holograms during simulated training. The green cube represents the important real-world object.

*Position Policy.* The policy to calculate the position to move the holograms was trained using reward function $R_1$, with various values for $\alpha$ and $\beta$, along with state space $S_1$ and action space $A_1$. The hyperparameters in table 7 were used along with the PPO algorithm.

*Direction Policy.* The policy to calculate the direction to move the holograms was trained using reward function $R_2$ with state space $S_2$ and action space $A_2$. The hyperparameters in table 8 were used along with the PPO algorithm.

### 3.5 Visual Output Security Policy Training Results

*Position Policy.* As illustrated in figure 15 left, training a RL model for the position policy with one hologram converged when not penalizing the agent for moving the hologram away from its original location (i.e. setting $\beta$ to 0 in $R_1$). However, the agent simply learned to move the hologram to the top right of the user's field of view in this case.

To overcome this problem, a RL agent for the position policy was also trained by penalizing the agent for moving the hologram away from its original location (i.e. setting $\beta$ to some positive number in $R_1$). Although various values of $\alpha$ and $\beta$ were tried, the model did not converge. An example of training with a penalty is shown in figure 15 right.

*Direction Policy.* As illustrated in figure 16, training a RL model for the direction policy with one hologram and three holograms both converged. However, the reward for the three hologram case was substantially lower. In addition, the holograms tended to simply move in a general direction (i.e. right) which led to the holograms moving away from their original locations more than necessary

| Parameter | Value |
|---|---|
| Number of Layers | 3 |
| Hidden Units | 128 |
| Batch Size | 64 |
| Beta | $5.0 \times 10^{-3}$ |
| Buffer Size | 2048 |
| Epsilon | 0.2 |
| Gamma | 0.99 |
| Lambda | 0.95 |
| Learning Rate | $3.0 \times 10^{-4}$ |
| Number of Epochs | 5 |
| Time Horizon | 2048 |
| Normalize | False |
| Use Recurrent | False |
| Use Curiosity | False |

Table 7. Hyperparameters used to train a RL agent to move holograms to a specific position away from an important real-world object.

| Parameter | Value |
|---|---|
| Number of Layers | 2 |
| Hidden Units | 64 |
| Batch Size | 10 |
| Beta | $1.0 \times 10^{-2}$ |
| Buffer Size | 512 |
| Epsilon | 0.2 |
| Gamma | 0.99 |
| Lambda | 0.95 |
| Learning Rate | $1.0 \times 10^{-3}$ |
| Number of Epochs | 5 |
| Time Horizon | 64 |
| Normalize | False |
| Use Recurrent | False |
| Use Curiosity | False |
| Max steps per training cycle | 100 to 500 |

Table 8. Hyperparameters used to train a RL agent to move holograms away from an important real-world object.

for them to no longer obscure the important real-world object. To overcome this issue the $R_3$ reward function was used for the final policy as discussed in the next section.

### 3.6 Deploying the Visual Output Security Policy

A full visual output security application was developed to test the deployment of a RL trained visual output security policy. The direction policy was chosen for this application since it performed

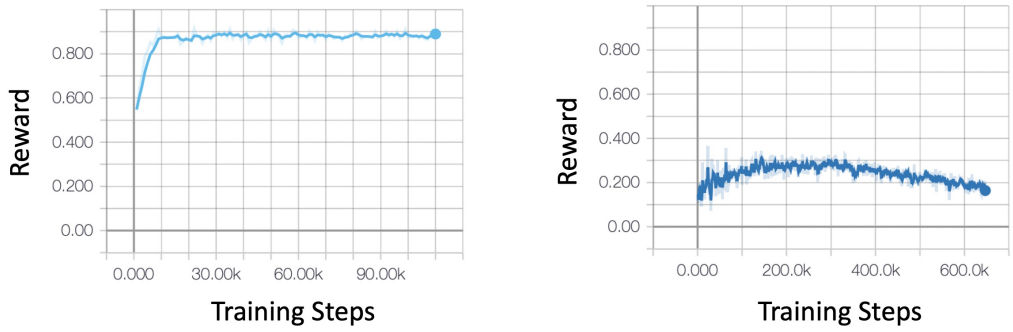


Fig. 15. **Left:** Training using the position policy with one hologram and no penalty for moving the hologram away from its original position. **Right:** Training using the position policy with one hologram and a penalty for moving the hologram away from its original position.

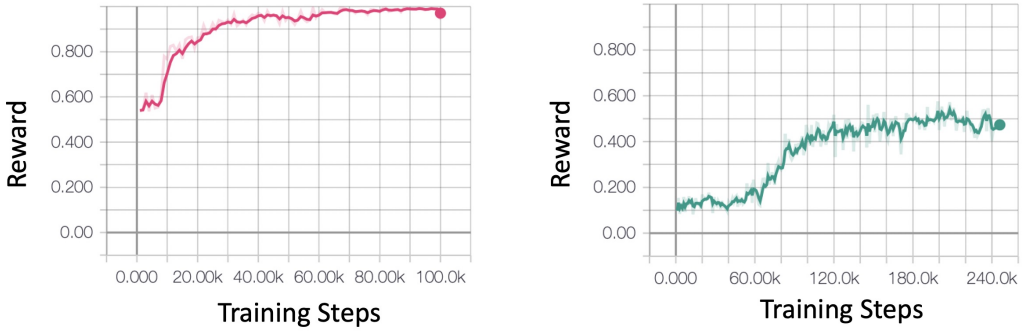


Fig. 16. **Left:** Training using the direction policy with one hologram. **Right:** Training using the direction policy with three holograms.

better than the position policy during initial testing. However, instead of training a model to move $N$ number of holograms away from an important real-world object, one model was trained using the $R_3$ reward function, $S_2$ state space and $A_2$ action space for a single hologram, enabling this single model to be used for an arbitrary number of holograms by controlling each hologram with its own agent. The modified $R_3$ reward function was used to incentivize the agent to move holograms away quickly by providing more reward the sooner the important real-world object was no longer obstructed and to keep the hologram in the same relative position to the important real-world object after moving the hologram by restricting the definition of the Success state.

For the purposes of this application, the important real-world object and hologram had a constant size and random location.

*Training.* The RL agent was trained using the hyperparameters in table 9. In addition, the distance from the important real-world object needed for the hologram to be considered no longer obstructing it was incrementally increased to help the model converge. Figure 17 shows the training of the direction policy RL agent.

*Image Tracking.* The image tracking library built into the Magic Leap One was used to recognize and track an important real-world object (an image of a stop sign). The user interface for the visual

| Parameter | Value |
|---|---|
| Number of Layers | 2 |
| Hidden Units | 128 |
| Batch Size | 10 |
| Beta | $5.0 \times 10^{-3}$ |
| Buffer Size | 100 |
| Epsilon | 0.2 |
| Gamma | 0.99 |
| Lambda | 0.95 |
| Learning Rate | $3.0 \times 10^{-4}$ |
| Number of Epochs | 3 |
| Time Horizon | 64 |
| Normalize | False |
| Use Recurrent | False |
| Use Curiosity | False |
| Max steps per training cycle | 100 to 500 |

Table 9. Hyperparameters used to train a RL agent to move holograms away from an important real-world object for the final deployment.

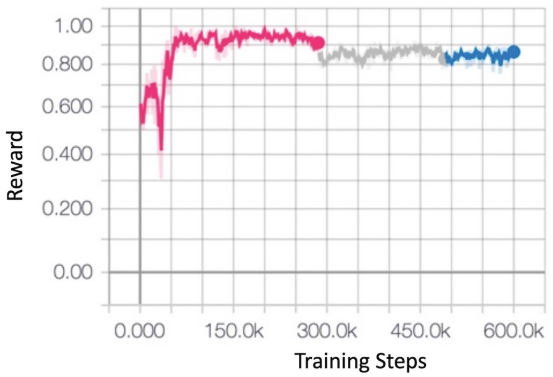


Fig. 17. Training using the direction policy with one hologram while incrementally increasing the distance of the hologram from the important real-world object in order to be in the Success state. Note that the varying line colors is a result of pausing then resuming training.

output security demonstration overlaid a green square on top of the important real-world object once it was recognized and tracked. Being able to apply the visual output security policy to more object classes requires the kind of semantic understanding which was investigated in section 2.

*Heuristics.* Some heuristics were used in addition to the RL trained direction visual output security policy:

- Only apply the RL trained direction policy if the hologram's original position obstructs the important real-world object.

- Once the RL trained direction policy is applied, move the hologram back to its original position as soon as its original position no longer obstructs the important real-world object.

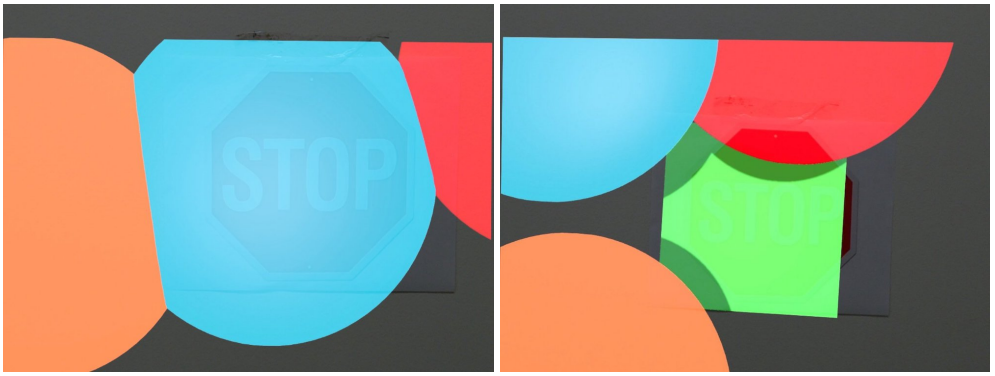

Fig. 18. **Left:** Example of what is displayed on the Magic Leap One display before the model-generated direction policy is applied to three holograms. **Right:** Example of what is displayed on the Magic Leap One display after the model-generated direction policy is applied to three holograms. The green square is overlaid in front of the important real-world object (a stop sign), indicating its position. In both cases, the user's field of view is actually larger but is clipped by the Magic Leap One screenshot functionality. In addition, the Magic Leap One screenshot also results in a slight misalignment of holograms.

*Deployment.* The visual output security application was deployed on a Magic Leap One without any noticeable performance degradation. Figure 18 shows an example of the user's field of view before the RL trained direction policy is applied (left) and after the policy is applied (right). A stop sign was used as the example important real-world object in this demonstration. As these images demonstrate, the application successfully identified an important real-world object and moved obstructing holograms. The identification and movement happen in real time.

### 3.7 Discussion

As the training of the position and direction policy using the $R_1$ and $R_2$ reward functions indicate, the reward function chosen has a direct effect on agent behavior. In the case of the position policy using $R_1$ with no penalty for moving the hologram away from its initial location, the model simply learned to always move the object to the top right corner which defeats the purpose of moving the obstructing hologram while keeping it as visible to the user as possible. Similarly, in the case of the direction policy using $R_2$ the agent learned to move the holograms in a general direction (i.e. right) which also moved the holograms away from its initial location. Furthermore, model convergence was an issue when training the position policy with the $R_1$ reward function with a penalty, which could potentially be overcome with more hyperparameter tuning.

While the convergence problem of the position policy may have been solved with more hyperparameter tuning, the approach of modifying the $R_2$ reward function to take into account the hologram's initial relative location to the important real-world object was used to arrive at a more scalable solution by using the $R_3$ reward function with one hologram. With this framework of training the RL agent for the visual output security problem, the final policy can be used by an arbitrary number of holograms, cutting back on the need to train a separate model for varying numbers of holograms. In addition, model convergence was aided by slowly increasing the distance the hologram has to move from the important real-world object to no longer obstruct it.

Although the built-in image tracking for the Magic Leap One was limited (the user had to be relatively close to the image and not too off center), the visual output security application functioned without any noticeable performance degradation for three holograms, indicating that a RL trained agent is a viable solution for visual output security.

# 4 SEMANTICALLY AWARE MESHING

## 4.1 Overview of Semantically Aware Meshing

Mesh generation is critical to AR because it provides information about surfaces in the user's environment. Using this surface information, AR applications can interact with the environment by, for example, placing an object on a table or "painting" a wall a different color by overlaying a colored surface on a wall. In particular, high quality meshes lead to a significant improvement in user experience. This section proposes a method to improve the mesh quality for AR applications using semantic cues gained from the work presented in section 2. Specifically, the proposal is to tune the priors used by mesh generation algorithms based on the semantic class of the points which are being meshed.

Take for example the task of rolling a virtual ball on a table. If the mesh of the table is imperfect, the ball may bounce around and change direction instead of smoothly rolling on the table as expected. Unfortunately, while the meshes generated by current AR devices are indeed impressive, they are still not of the highest quality. Sometimes meshes are incomplete and objects fall through surfaces. In other cases meshes are not smooth and lead to inaccurate physics.

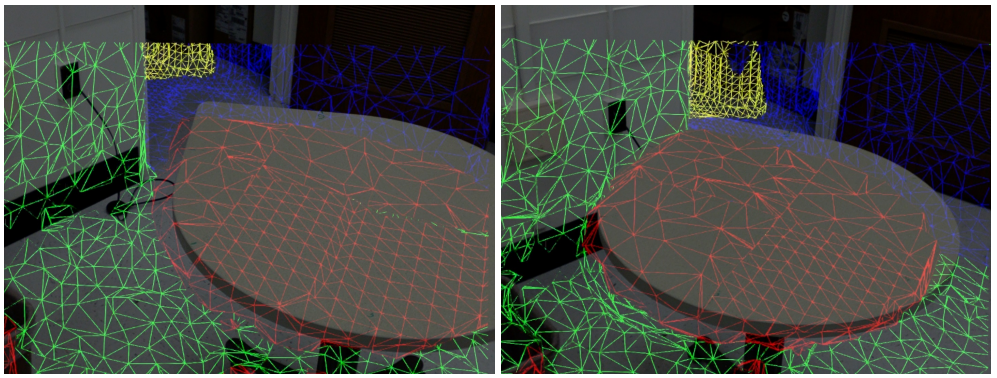

Fig. 19. Examples of meshes generated from the Magic Leap One. There is some jaggedness of the mesh for the table even though it is smooth, although it may be difficult to perceive in the screenshot due to the lack of depth cues. Note that the meshes are slightly misaligned due to limitations in the Magic Leap One's screenshot functionality.

Figure 19 shows an example of a mesh generated by the Magic Leap One. The Magic Leap One has very good localization which allows one to view the mesh from different angles with minimal drift. However, as illustrated by figure 19 the meshes generated by the Magic Leap One can be jagged even for smooth surfaces. In addition, mesh quality degrades if there are minor perturbations, such as a pencil on a table. Mesh quality does improve the longer one points the Magic Leap One sensor at an object and also by moving around an object. Nonetheless, an ideal user experience would be to provide high quality meshes without needing to diligently map an environment.

One way to improve the mesh quality for AR applications may be to incorporate semantic information into the meshing process. Using the kind of semantic understanding of the environment

gained from section 2, we could provide semantic cues to meshing algorithms to improve their quality. For example, if we know a set of point cloud points belong to a table, the mesh generation algorithm can be tuned to ensure smoothness of the meshed surface. Further, using semantic cues may also allow the algorithm to focus on meshing important details. For example, an AR application may require a high quality mesh of a table because objects are placed on it but not a chair because the application does not interact with chairs. In this way, computational resources can be delegated to improving the mesh for objects which make the most impact on a user's experience.

### 4.2 Related Work

Mesh generation is itself a well studied field. Here, mesh generation from point cloud data will be considered with a focus on previous work in mesh generation which has potential to be aided by semantic cues.

In general, mesh generation algorithms have a set of priors. Such priors are used by mesh generation algorithms to tackle imperfections in the point cloud data by taking into account properties of the point cloud such as noise, misalignment, and density as well as assumptions about the object being meshed such as smoothness and shape primitives [7].

A meshing algorithm may require information about surface normals. Normals are the direction perpendicular to the local surface approximation for each point. Such normals can be obtained using various methods [10, 25, 49]. Once these normals are obtained, they are useful for surface reconstruction algorithms [9, 32]. For example, once one has an oriented normal for each point, the zero crossings of the normals can be inferred to be the surfaces [7].

There are two types of mesh generation priors which may have the potential to be tuned by semantic cues. The first is smoothness which constrains the reconstructed surface to a desired level of smoothness [7]. In particular, local smoothness is used to combat issues of insufficient or missing samples [3, 7, 25], whereas global smoothness is used in the reconstruction of objects to produce watertight surfaces [7, 9, 32]. Another useful kind of prior is geometric primitives which assumes that the geometry of the object being meshed can be described by simple geometric shapes such as planes, spheres, cylinders, and cubes and are considered useful for meshing indoor scenes [7].

Another interesting prior used in mesh generation is data driven approaches. Data driven approaches use a catalog of 3D models and match objects in the scene to the most appropriate model [7]. While this method is useful for scene reconstruction [34, 47, 61] it does not provide a true mesh of the scene since the objects selected for an object class (e.g. chair) may not be the exact same chair as the one which is being meshed.

Some recent work has applied deep learning to mesh generation. Li et al. propose Supervised Primitive Fitting Network [39] which is a model that can detect a number of geometric primitives to generate a mesh and utilizes a PointNet++ [52] backbone. Kanazawa et al. have presented a model which can predict the 3D texture and shape of an object from a single RGB image [31], which is radically different from how meshes are currently generated using RGB-D data.

### 4.3 Semantically Aware Meshing: A Proposal

The proposal to use semantic cues for aiding mesh generation is centered on three ideas:

- Use semantic information to adjust the smoothness prior of mesh generation algorithms.
- Choose geometric primitives for regions of a point cloud based on semantic cues during mesh generation.
- Reduce the focus of generating high quality meshes on regions which are not essential to the user experience or for regions which cannot be meshed accurately.

While current meshing algorithms in use for AR devices can generate high quality meshes given enough time and data points, the idea of semantically aware meshing is to provide the same quality, or better, meshes faster and without as much data. For example, take the case of meshing the surface of a table. Given a set of points in the point cloud which are labeled as table surface, we could apply a plane as a geometric primitive to mesh the region. The result would be a smooth surface which interacts with AR objects in a manner that the user expects. For other objects, such as a chair, which may be harder to describe using just geometric primitives we can tune smoothness parameters for each object to generate high quality meshes. Further, some objects such as a tree simply cannot be meshed well with current meshing algorithms. This is because it is difficult to distinguish each leaf and branch given a point cloud. For such objects the semantically aware meshing algorithm can "give up" and not exert unnecessary compute on trying to refine the mesh.

Although deep learning methods may provide higher quality meshes as algorithms improve and more data sets are made available, it is likely that such models would not be real-time, especially given the constraints of an AR deployment. Therefore, it is advantageous to use the semantic information which various AR applications could also benefit from and use it as a prior for classical meshing algorithms which can produce high quality meshes.

## 5 FUTURE WORK

*Semantic Understanding.* While the semantic understanding pipeline described in section 2 presents a framework on which to build AR experiences that exploit user context, there is still much to be done to improve the quality and detail of the semantic model as well as its real-time performance. The ultimate goal would be to fuse one or more semantic models together with SLAM algorithms to build a semantic map of the world that contains both static and dynamic content. Such a detailed semantic map would allow for truly immersive and persistent AR experiences as virtual objects would retain their physical location across interactions.

*Visual Output Security.* Further research must be done on how to decide what constitutes an important real-world object. Such criteria may change depending on the user's context (e.g. driving versus reading). In addition, more robust models for visual output security could be developed using techniques such as curriculum learning and training agents to cooperate with each other, and add parameters such that holograms do not obstruct each other while staying as close to their original position as possible. Moreover, future AR output security work may also consider non-visual output such as audio and haptic output.

*Semantically Aware Meshing.* A meshing pipeline which utilizes semantic cues to aid in the fine tuning of priors should be developed and benchmarked against other methods. Such a method will hopefully improve mesh quality while maintaining real-time performance on AR devices. In addition, deep learning models which can be executed on raw point clouds may be a promising method of improving mesh quality, although they may not have real-time guarantees. Mesh quality will need to be balanced with sensor quality and algorithm latency for AR deployments.

## 6 CONCLUSION

Semantic understanding has the potential to greatly improve AR experiences. The work done in this project presents a proof-of-concept of an end-to-end system which provides semantic context for an AR application using a 2D semantic segmentation model and data from a RGB-D camera. Moreover, this system provides the foundation on which AR applications that exploit semantic cues can be built. Unless mobile compute improves and model complexity is reduced such that real-time inference of semantic models is possible directly on AR devices, a real-world deployment

of AR applications which need a rich understanding of the environment will most likely require edge servers, much like what was used in this study.

Visual output security was presented and investigated as one application which can benefit greatly from semantic context. The work developed a proof of concept AR visual output security application using a RL trained policy, demonstrating that RL trained models are a viable method for developing policies for output security. The demonstration also showed that RL models can be executed on a physical AR device, opening the doors for applying RL to more problems in AR.

Finally, semantically aware meshing was proposed as a method to improve the quality of meshes for AR applications. Because meshes are vital to the user experience in AR, improving their quality is of interest to enable high quality AR applications.

As AR improves and the line between the physical and virtual worlds blur, AR devices will need to be smarter to enable the user to seamlessly interact with both virtual and physical objects. An understanding of the user's environment is vital to enable such rich experiences and there is still much work to be done to bring about the kind of fidelity and quality of understanding necessary for a truly immersive experience. Nonetheless, the foundation of a semantic understanding pipeline for a physical AR device and the exploration of visual output security and semantically aware meshing provides a framework on which to build more seamless AR.

## ACKNOWLEDGMENTS

This work is supported in part by NSF grants CSR-1903136 and CNS-1908051, and by the Lord Foundation of North Carolina. The SIGMOBILE Travel Grant and Duke URS Travel grant provided funding to present part of this work at ACM SenSys '19. I would also like to thank Surin Ahn for her insight into reinforcement learning applied to visual output security, Tim Scargill for his help in understanding the Magic Leap One's meshing capabilities, and Professor Gorlatova for her guidance.

## REFERENCES

[1] Surin Ahn, Maria Gorlatova, Parinaz Naghizadeh, and Mung Chiang. 2019. Personalized Augmented Reality via Fog-Based Imitation Learning. In *Proceedings of the Workshop on Fog Computing and the IoT* (Montreal, Quebec, Canada) *(IoT-Fog '19)*. Association for Computing Machinery, New York, NY, USA, 11–15. https://doi.org/10.1145/3313150.3313219

[2] Surin Ahn, Maria Gorlatova, Parinaz Naghizadeh, Mung Chiang, and Prateek Mittal. 2018. Adaptive Fog-Based Output Security for Augmented Reality. In *Proceedings of the 2018 Morning Workshop on Virtual Reality and Augmented Reality Network* (Budapest, Hungary) *(VR/AR Network '18)*. Association for Computing Machinery, New York, NY, USA, 1–6. https://doi.org/10.1145/3229625.3229626

[3] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. 2003. Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics* 9, 1 (2003), 3–15.

[4] Apple. [n.d.]. *ARKit*. Retrieved Spring, 2020 from https://developer.apple.com/augmented-reality/

[5] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. 2017. Deep Reinforcement Learning: A Brief Survey. *IEEE Signal Processing Magazine* 34, 6 (Nov 2017), 26–38. https://doi.org/10.1109/msp.2017.2743240

[6] V. Badrinarayanan, A. Kendall, and R. Cipolla. 2017. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39, 12 (Dec 2017), 2481–2495. https://doi.org/10.1109/TPAMI.2016.2644615

[7] Matthew Berger, Andrea Tagliasacchi, Lee M. Seversky, Pierre Alliez, Gaël Guennebaud, Joshua A. Levine, Andrei Sharf, and Claudio T. Silva. 2017. A Survey of Surface Reconstruction from Point Clouds. *Comput. Graph. Forum* 36, 1 (Jan. 2017), 301–329. https://doi.org/10.1111/cgf.12802

[8] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard. 2016. Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age. *IEEE Transactions on Robotics* 32, 6 (Dec 2016), 1309–1332. https://doi.org/10.1109/TRO.2016.2624754

[9] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. 2001. Reconstruction and Representation of 3D Objects with Radial Basis Functions. In *Proceedings of the 28th Annual Conference on Computer*

*Graphics and Interactive Techniques (SIGGRAPH '01)*. Association for Computing Machinery, New York, NY, USA, 67–76. https://doi.org/10.1145/383259.383266

[10] F. Cazals and M. Pouget. 2003. Estimating Differential Quantities Using Polynomial Fitting of Osculating Jets. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing* (Aachen, Germany) *(SGP '03)*. Eurographics Association, Goslar, DEU, 177–187.

[11] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas. 2017. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 77–85.

[12] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, and R. Urtasun. 2016. Monocular 3D Object Detection for Autonomous Driving. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2147–2156.

[13] Zhuo Chen, Wenlu Hu, Junjue Wang, Siyan Zhao, Brandon Amos, Guanhang Wu, Kiryong Ha, Khalid Elgazzar, Padmanabhan Pillai, Roberta Klatzky, Daniel Siewiorek, and Mahadev Satyanarayanan. 2017. An Empirical Study of Latency in an Emerging Class of Edge Computing Applications for Wearable Cognitive Assistance. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing* (San Jose, California) *(SEC '17)*. Association for Computing Machinery, New York, NY, USA, Article 14, 14 pages. https://doi.org/10.1145/3132211.3134458

[14] Open AR Cloud. 2019. *State of The AR Cloud Report*. Retrieved Spring, 2020 from https://www.openarcloud.org/reports/sotarc-2019

[15] Joseph DeChicchis. [n.d.]. *Dataset and Model on GitHub*. Retrieved Spring, 2020 from https://github.com/jdechicchis/ar-semantic-understanding-data

[16] Joseph DeChicchis. [n.d.]. *Project Code on GitHub*. Retrieved Spring, 2020 from https://github.com/jdechicchis/ar-semantic-understanding

[17] Joseph DeChicchis, Surin Ahn, and Maria Gorlatova. 2019. Adaptive AR Visual Output Security Using Reinforcement Learning Trained Policies: Demo Abstract. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems* (New York, New York) *(SenSys '19)*. Association for Computing Machinery, New York, NY, USA, 380–381. https://doi.org/10.1145/3356250.3361935

[18] David Eigen and Rob Fergus. 2015. Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-Scale Convolutional Architecture. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV) (ICCV '15)*. IEEE Computer Society, USA, 2650–2658. https://doi.org/10.1109/ICCV.2015.304

[19] Facebook. 2018. *F8 2018: Augmented Reality Comes to Messenger*. Retrieved Spring, 2020 from https://www.facebook.com/business/news/f8-2018-augmented-reality-comes-to-messenger

[20] R. Girshick. 2015. Fast R-CNN. In *2015 IEEE International Conference on Computer Vision (ICCV)*. 1440–1448.

[21] R. Girshick, J. Donahue, T. Darrell, and J. Malik. 2014. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 580–587.

[22] Google. [n.d.]. *ARCore*. Retrieved Spring, 2020 from https://developers.google.com/ar/

[23] K. He, X. Zhang, S. Ren, and J. Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778. https://doi.org/10.1109/CVPR.2016.90

[24] HERE. [n.d.]. *HERE HD Live Map*. Retrieved Spring, 2020 from https://www.here.com/products/automotive/hd-maps

[25] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. 1992. Surface Reconstruction from Unorganized Points. *SIGGRAPH Comput. Graph.* 26, 2 (July 1992), 71–78. https://doi.org/10.1145/142920.134011

[26] X. Hou, Y. Lu, and S. Dey. 2017. Wireless VR/AR with Edge/Cloud Computing. In *2017 26th International Conference on Computer Communication and Networks (ICCCN)*. 1–8.

[27] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. https://arxiv.org/abs/1704.04861

[28] Intel. [n.d.]. *Intel RealSense*. Retrieved Spring, 2020 from https://www.intelrealsense.com/

[29] Suman Jana, David Molnar, Alexander Moshchuk, Alan Dunn, Benjamin Livshits, Helen J. Wang, and Eyal Ofek. 2013. Enabling Fine-Grained Permissions for Augmented Reality Applications with Recognizers. In *22nd USENIX Security Symposium (USENIX Security 13)*. USENIX Association, Washington, D.C., 415–430. https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/jana

[30] A. Janoch, S. Karayev, Yangqing Jia, J. T. Barron, M. Fritz, K. Saenko, and T. Darrell. 2011. A category-level 3-D object dataset: Putting the Kinect to work. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*. 1168–1174. https://doi.org/10.1109/ICCVW.2011.6130382

[31] Angjoo Kanazawa, Shubham Tulsiani, Alexei A. Efros, and Jitendra Malik. 2018. Learning Category-Specific Mesh Reconstruction from Image Collections. https://arxiv.org/abs/1803.07549

[32] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. 2006. Poisson Surface Reconstruction. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing* (Cagliari, Sardinia, Italy) *(SGP '06)*. Eurographics Association, Goslar, DEU, 61–70.

[33] B. Kim, P. Kohli, and S. Savarese. 2013. 3D Scene Understanding by Voxel-CRF. In *2013 IEEE International Conference on Computer Vision*. 1425–1432. https://doi.org/10.1109/ICCV.2013.180

[34] Young Min Kim, Niloy J. Mitra, Dong-Ming Yan, and Leonidas Guibas. 2012. Acquiring 3D Indoor Environments with Variability and Repetition. *ACM Trans. Graph.* 31, 6, Article 138 (Nov. 2012), 11 pages. https://doi.org/10.1145/2366145.2366157

[35] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet Classification with Deep Convolutional Neural Networks. *Commun. ACM* 60, 6 (May 2017), 84–90. https://doi.org/10.1145/3065386

[36] Magic Leap. [n.d.]. *Magic Leap One.* Retrieved Spring, 2020 from https://www.magicleap.com/magic-leap-one

[37] Kiron Lebeck, Tadayoshi Kohno, and Franziska Roesner. 2016. How to Safely Augment Reality: Challenges and Directions. In *Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications* (St. Augustine, Florida, USA) *(HotMobile '16)*. Association for Computing Machinery, New York, NY, USA, 45–50. https://doi.org/10.1145/2873587.2873595

[38] K. Lebeck, K. Ruth, T. Kohno, and F. Roesner. 2017. Securing Augmented Reality Output. In *2017 IEEE Symposium on Security and Privacy (SP)*. 320–337. https://doi.org/10.1109/SP.2017.13

[39] L. Li, M. Sung, A. Dubrovina, L. Yi, and L. J. Guibas. 2019. Supervised Fitting of Geometric Primitives to 3D Point Clouds. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2647–2655. https://doi.org/10.1109/CVPR.2019.00276

[40] L. Liu, J. Lu, C. Xu, Q. Tian, and J. Zhou. 2019. Deep Fitting Degree Scoring Network for Monocular 3D Object Detection. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 1057–1066.

[41] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. 2016. SSD: Single Shot MultiBox Detector. *Lecture Notes in Computer Science* (2016), 21–37. https://doi.org/10.1007/978-3-319-46448-0_2

[42] J. Long, E. Shelhamer, and T. Darrell. 2015. Fully convolutional networks for semantic segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 3431–3440.

[43] Microsoft. [n.d.]. *HoloLens.* Retrieved Spring, 2020 from https://www.microsoft.com/en-us/hololens

[44] Microsoft. 2018. *What is mixed reality?* Retrieved Spring, 2020 from https://docs.microsoft.com/en-us/windows/mixed-reality/mixed-reality

[45] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533. https://doi.org/10.1038/nature14236

[46] Kaichun Mo, Shilin Zhu, Angel X. Chang, Li Yi, Subarna Tripathi, Leonidas J. Guibas, and Hao Su. 2018. PartNet: A Large-Scale Benchmark for Fine-Grained and Hierarchical Part-Level 3D Object Understanding. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2018), 909–918.

[47] Liangliang Nan, Ke Xie, and Andrei Sharf. 2012. A Search-Classify Approach for Cluttered Indoor Scene Understanding. *ACM Trans. Graph.* 31, 6, Article 137 (Nov. 2012), 10 pages. https://doi.org/10.1145/2366145.2366156

[48] D. Pannen, M. Liebner, and W. Burgard. 2019. HD Map Change Detection with a Boosted Particle Filter. In *2019 International Conference on Robotics and Automation (ICRA)*. 2561–2567. https://doi.org/10.1109/ICRA.2019.8794329

[49] Mark Pauly, Niloy J. Mitra, and Leonidas J. Guibas. 2004. Uncertainty and Variability in Point Cloud Surface Data. In *Proceedings of the First Eurographics Conference on Point-Based Graphics* (Switzerland) *(SPBG'04)*. Eurographics Association, Goslar, DEU, 77–84.

[50] Q. Pham, T. Nguyen, B. Hua, G. Roig, and S. Yeung. 2019. JSIS3D: Joint Semantic-Instance Segmentation of 3D Point Clouds With Multi-Task Pointwise Networks and Multi-Value Conditional Random Fields. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 8819–8828.

[51] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas. 2018. Frustum PointNets for 3D Object Detection from RGB-D Data. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 918–927.

[52] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. 2017. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 5105–5114.

[53] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. 2016. You Only Look Once: Unified, Real-Time Object Detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 779–788. https://doi.org/10.1109/CVPR.2016.91

[54] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.). Curran Associates, Inc., 91–99. http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.pdf

[55] Franziska Roesner, Tadayoshi Kohno, and David Molnar. 2014. Security and Privacy for Augmented Reality Systems. *Commun. ACM* 57, 4 (April 2014), 88–96. https://doi.org/10.1145/2580723.2580730

[56] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. https://arxiv.org/abs/1505.04597

[57] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen. 2018. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 4510–4520. https://doi.org/10.1109/CVPR.2018.00474

[58] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. 2009. The Case for VM-Based Cloudlets in Mobile Computing. *IEEE Pervasive Computing* 8, 4 (2009), 14–23.

[59] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. 2015. High-Dimensional Continuous Control Using Generalized Advantage Estimation. https://arxiv.org/abs/1506.02438

[60] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. https://arxiv.org/abs/1707.06347

[61] Tianjia Shao, Weiwei Xu, Kun Zhou, Jingdong Wang, Dongping Li, and Baining Guo. 2012. An Interactive Approach to Semantic Modeling of Indoor Scenes with an RGBD Camera. *ACM Trans. Graph.* 31, 6, Article 136 (Nov. 2012), 11 pages. https://doi.org/10.1145/2366145.2366155

[62] S. Shi, X. Wang, and H. Li. 2019. PointRCNN: 3D Object Proposal Generation and Detection From Point Cloud. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–779.

[63] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. 2012. Indoor Segmentation and Support Inference from RGBD Images. In *Proceedings of the 12th European Conference on Computer Vision - Volume Part V* (Florence, Italy) *(ECCV'12)*. Springer-Verlag, Berlin, Heidelberg, 746–760. https://doi.org/10.1007/978-3-642-33715-4_54

[64] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489. https://doi.org/10.1038/nature16961

[65] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1409.1556

[66] Snapchat. [n.d.]. *Lens Studio.* Retrieved Spring, 2020 from https://lensstudio.snapchat.com/

[67] S. Song, S. P. Lichtenberg, and J. Xiao. 2015. SUN RGB-D: A RGB-D scene understanding benchmark suite. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 567–576. https://doi.org/10.1109/CVPR.2015.7298655

[68] Christopher Streiffer, Animesh Srivastava, Victor Orlikowski, Yesenia Velasco, Vincentius Martin, Nisarg Raval, Ashwin Machanavajjhala, and Landon P. Cox. 2017. EPrivateeye: To the Edge and Beyond!. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing* (San Jose, California) *(SEC '17)*. Association for Computing Machinery, New York, NY, USA, Article 18, 13 pages. https://doi.org/10.1145/3132211.3134457

[69] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction* (second ed.). The MIT Press.

[70] Mingxing Tan and Quoc V. Le. 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA (Proceedings of Machine Learning Research)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.), Vol. 97. PMLR, 6105–6114. http://proceedings.mlr.press/v97/tan19a.html

[71] TechRadar. 2019. *Apple AR glasses release date, news and rumors.* Retrieved Spring, 2020 from https://www.techradar.com/news/apple-ar-glasses-release-date-news-and-rumors

[72] TomTom. [n.d.]. *TomTom HD Map.* Retrieved Spring, 2020 from https://www.tomtom.com/products/hd-map/

[73] J. R. Uijlings, K. E. Sande, T. Gevers, and A. W. Smeulders. 2013. Selective Search for Object Recognition. *Int. J. Comput. Vision* 104, 2 (Sept. 2013), 154–171. https://doi.org/10.1007/s11263-013-0620-5

[74] Unity. [n.d.]. *ml-agents.* Retrieved Spring, 2020 from https://github.com/Unity-Technologies/ml-agents

[75] Unity. [n.d.]. *Unity.* Retrieved Spring, 2020 from https://unity.com/

[76] V. Vineet, O. Miksik, M. Lidegaard, M. Nießner, S. Golodetz, V. A. Prisacariu, O. Kähler, D. W. Murray, S. Izadi, P. Pérez, and P. H. S. Torr. 2015. Incremental dense semantic stereo fusion for large-scale semantic scene reconstruction. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 75–82.

[77] W. Wu, Z. Qi, and L. Fuxin. 2019. PointConv: Deep Convolutional Networks on 3D Point Clouds. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 9613–9622.

[78] Zifeng Wu, Chunhua Shen, and Anton van den Hengel. 2016. Wider or Deeper: Revisiting the ResNet Model for Visual Recognition. https://arxiv.org/abs/1611.10080

[79] J. Xiao, A. Owens, and A. Torralba. 2013. SUN3D: A Database of Big Spaces Reconstructed Using SfM and Object Labels. In *2013 IEEE International Conference on Computer Vision*. 1625–1632. https://doi.org/10.1109/ICCV.2013.458

[80] Hengshuang Zhao, Xiaojuan Qi, Xiaoyong Shen, Jianping Shi, and Jiaya Jia. 2017. ICNet for Real-Time Semantic Segmentation on High-Resolution Images. https://arxiv.org/abs/1704.08545

[81] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia. 2017. Pyramid Scene Parsing Network. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 6230–6239.