ODDS: Real-Time Object Detection using Depth Sensors on Embedded GPUs

Niluthpol Chowdhury Mithun
University of California, Riverside, CA
nmith001@ucr.edu

Sirajum Munir
Bosch Research and Technology Center, PA
sirajum.munir@us.bosch.com

Karen Guo
University of Minnesota, MN
guoxx431@umn.edu

Charles Shelton
Bosch Research and Technology Center, PA
charles.shelton@us.bosch.com

ABSTRACT

Detecting objects that are carried when someone enters or exits a room is very useful for a wide range of smart building applications including safety, security, and energy efficiency. While there has been a significant amount of work on object recognition using large-scale RGB image datasets, RGB cameras are too privacy invasive in many smart building applications and they work poorly in the dark. Additionally, deep object detection networks require powerful and expensive GPUs. We propose a novel system that we call ODDS (Object Detector using a Depth Sensor) that can detect objects in real-time using only raw depth data on an embedded GPU, e.g., NVIDIA Jetson TX1. Hence, our solution is significantly less privacy invasive (even if the sensor is compromised) and less expensive, while maintaining a comparable accuracy with state of the art solutions. Specifically, we resort to training a deep convolutional neural network using raw depth images, with curriculum based learning to improve accuracy by considering the complexity and imbalance in object classes and developing a sparse coding based technique that speeds up the system ~2x with minimal loss of accuracy. Based on a complete implementation and real-world evaluation, we see ODDS achieve 80.14% mean average precision in object detection in real-time (5-6 FPS) on a Jetson TX1.

CCS CONCEPTS

• **Computer systems organization** → *Embedded and cyber-physical systems*; *Embedded systems*; *Real-time system architecture*; • **Hardware** → *Sensor applications and deployments*;

KEYWORDS

Object detection, Embedded systems, Deep learning, Curriculum learning, Network pruning

ACM Reference format:
Niluthpol Chowdhury Mithun, Sirajum Munir, Karen Guo, and Charles Shelton. 2017. ODDS: Real-Time Object Detection using Depth Sensors on Embedded GPUs. In *Proceedings of IPSN '18, Porto, Portugal, April 11–13, 2018,* 12 pages.

https://doi.org/10.1145/3137133.3141461

1 INTRODUCTION

Detecting humans and objects that are carried inside and outside of rooms in buildings is very useful for a wide range of smart building applications including safety, security, and energy efficiency. For example, if it is detected that someone is entering a room with a box and the box is not taken back, it could be a security threat. As gun violence is on the rise in the US, if it is detected that someone is entering with a gun, the neighboring people and local police can be alerted. Also, it can be useful for reminding people if someone enters a room with an umbrella or a laptop and forgets to take it back when leaving (e.g., by playing a chime). It is also useful for controlling Heating Ventilation and Air Conditioning (HVAC) systems to save energy. For example, in a commercial space, if someone is leaving his office with a backpack in the afternoon, it may mean he is leaving for the day, whereas if he is leaving with a laptop, it may mean he is going out for a meeting, and leaving empty handed may mean that he is leaving for a restroom/lunch break. This additional contextual information can be useful for controlling HVAC systems efficiently.

There has been a significant amount of work on image recognition/classification (AlexNet [29], GoogleNet [49]) and object detection (SSD [32], FasterRCNN [43]). Given an image, the former one attempts to classify the image to a single/multiple categories, whereas the latter does so in addition to localizing the objects and hence the latter one is more challenging. Object detection solutions can detect multiple objects of the same category (and their positions) in an image that can not be done by image classification techniques. Existing solutions for object detection are mainly based on RGB images. However, RGB camera based solutions may not be suitable for many smart building applications as RGB cameras are too privacy invasive to be deployed in many settings, e.g., in office rooms, and they work poorly in low illumination environments. The solutions that use depth sensors for object detection either use RGB data as complementary to the depth data [19] or use the corresponding RGB data tofi lter out noise in the depth data [20]. Our solution is based only on depth sensor data as we feed noisy depth data directly into a CNN and hence it saves the cost of a corresponding RGB camera. Also, as a result, it is much less privacy invasive (even if the sensor is compromised) and works in a completely dark environment.

Depth sensors bring additional challenges to generic object detection. For example, many pre-trained RGB image classification networks are available with excellent generalization ability and

0-7695-6377-5/18/$31.00 ©2018 IEEE
DOI 10.1109/IPSN.2018.00051

almost all RGB based object detection networks are built on top of a pre-trained RGB image classification network utilizing transfer learning. Transfer learning is very common in practice, as training a deep CNN from scratch needs a dataset of a sufficiently large size, which is rare for most tasks. Hence, most of the RGB based CNN networks use a previously trained CNN (trained on ImageNet, which has millions of images) on large dataset as initialization and thenfi netune the network on the new task. However, the ability to transfer decreases significantly as the distance between task increases [55]. There is hardly any raw depth image based pre-trained network available, which makes it very difficult to train deep CNN based depth image based object detection network for our task. Significant noise in depth images and limited availability of depth images on the web makes training even more difficult. Also, most proposed solutions for object detection require the use of powerful and expensive GPUs [43], e.g., Nvidia GTX 1080 Ti costs ~$750 and K40 costs ~$3000 whereas we perform the entire processing on a cheaper embedded GPU Jetson TX1 (the development kit costs ~$500 which includes a TX1 module in a computer). This is very challenging as embedded GPUs are extremely weak in terms of computation power. As an example, when porting an object detection solution from a Nvidia GTX 1080 Ti to a TX1, we see a drop of frame rate from 69.33 FPS to 5.9 FPS.

Our proposed system is called ODDS (Object Detector using a Depth Sensor). It requires mounting of a depth sensor at a doorway looking downwards as shown in Figure 1. Mounting in such a way helps to avoid occlusion a great extent. For prototyping, we use the depth sensor of Microsoft Kinect for Xbox One. However, we use raw depth data and hence other depth sensors with similar functionality will work. We use an existing object detection network (SSD [32]) that is built for RGB images as a tool and show how transfer learning can be used on a different sensing modality (depth). A depth frame contains so much noise and it has so low resolution that it is a fundamental question if objects that are a few feet away from the sensor can be detected using it on an embedded GPU in real-time. We address several realistic challenges to develop ODDS, e.g., addressing class imbalance issue, determining the order of samples for training to improve accuracy, determining techniques to speed up to be able to run on an embedded GPU, and show our techniques work in a completely different sensing modality (depth). We train a deep convolutional network (CNN) model based on SSD architecture for our task. To guide the model to achieve higher accuracy, we design a curriculum learning path for training by utilizing our prior knowledge of training data as the order and selection of training data helps the model to reach a better local minimum. To improve speed, we propose a sparse coding based filter-level network pruning algorithm that concurrently speeds up and compresses CNN models.

This work has the following contributions. First, we design and implement a novel system called ODDS, which is thefi rst system that detects various types of objects including backpacks, boxes, laptops, phones, umbrellas, cups, nerf guns (as real guns are not allowed in the office), and humans in smart buildings using raw noisy depth images, without using the corresponding RGB image to remove noise as in state of the art solutions. Instead of developing another heavyweight generic object detection solution requiring extensive computation, we only consider a handful of classes relevant to smart buildings, more specifically commercial and academic buildings, and ODDS is thefi rst system that can detect the aforementioned objects on an embedded GPU e.g., NVidia Jetson TX1 in real-time. Second, we employ novel strategies to improve the accuracy and speed of ODDS. In order to improve accuracy, inspired by the learning process of humans and animals, we leverage curriculum learning to train ODDS to detect and learn objects from simple to complex shapes instead of feeding the training samples altogether. In order to speed up, we introduce sparse coding based representativefi lter selection using group $l_{2,1}$ norm which can be used with any off-the-shelf deep learning libraries, in contrast to most existing approaches that require changing the implementation of underlying frameworks, e.g., Caffe or TensorFlow. Third, we collect a 21 GB dataset from a commercial space containing 200K depth frames. Among these frames, around 65K depth images contain meaningful objects, where we annotate the positions of each object (mentioned above). To the best of our knowledge, no depth dataset exists for these objects where the depth sensor is mounted at the ceiling looking downwards and we make the dataset available in [4]. Fourth, based on extensive evaluation using over 13000 depth frames, we see ODDS achieve 80.14% mean average precision in real-time (5-6) FPS on Jetson TX1. We also benchmark the performance in two powerful GPUs (GTX 1080 Ti, K40) and another embedded GPU (Jetson TK1).

Figure 1: Placement of a depth sensor (Kinect for XBOX One) at the ceiling near a doorway.

2 RELATED WORK

Image Classification Methods: Methods for image classification can be divided into two main categories, e.g., hand-designed feature extractors and feature learning. Hand-designed features can be further divided into local features and global features. SIFT [33], HOG [15] and SURF [10] are all examples of local feature descriptors that describe the salient regions around key points in an image. Global image descriptors such as the color histogram [51], color coherence vector [44], and Gist [45] describe the image as a whole. After extracting features from images, machine learning is used to

map the features to categories [52]. More recently, however, feature extraction methods involve learning visual representations directly from pixels using deep neural networks [29, 31], e.g., AlexNet [29], GoogleNet, and ResNet. These deep CNNs have been trained on millions of images and hence, they are highly capable of capturing generic features from images. These deep CNN features are now the state-of-the-art image features and show remarkable performance in most recognition tasks.

Object Detection Methods: Object detection is a more challenging task than image classification. Unlike image classification, which requires labeling an image to a particular category, detection requires localizing (likely many) objects within an image and assigning each object a label. As the training data for object detection is costly, the object detection networks usually utilize CNNs trained on image classification task to start with. It has been shown that a CNN trained on image classification task can lead to dramatically higher object detection performance as compared to systems based on hand-crafted features. The deep CNNs trained on image classification task have been used as a starting block by almost all state of the art deep network based object detection approaches for training a high-capacity model as the quantity of annotated detection data is limited.

Similar to image classification, methods for object detection can be divided into two main categories: hand-designed feature extractors and feature learning. In [19], authors detect contours on depth images for segmentation and quantize the outputs as features for classifying a scene. In [9], authors use quantized local features for detecting and segmenting objects. Current state-of-the-art relies on transferring andfi ne tuning CNN trained on RGB to depth data [20, 48, 59]. In [47], a CNN is trained on patches in an unsupervised way and combined with a recurrent neural network for RGB-D object classification. In [20], Region-CNN has been used to detect objects from depth images. In [20, 59], authors propose tofine-tune CNN jointly based on RGB and depth image pairs. In [59], authors jointlyfi ne-tune both the RGB and depth CNN utilizing a multi-modal fusion layer which considers both inter and intra-modal correlations. Alternatively, transferring RGB CNN model to the depth CNN based on RGB-depth image pairs was proposed in [20].

Curriculum Learning: Curriculum learning is a training strategy [11, 41], where the complexity of training data is taken into account in training by distributing/dividing training samples based on complexity. Based on this strategy, network training usually starts on simple tasks/examples and training is done progressively on more difficult tasks/examples. Curriculum learning has been shown to be effective in guiding the training process to achieve better performance in different tasks [25]. Adding a pre-defined learning curriculum [25] to the training can take into account the helpful knowledge known before training and improve model performance. Inspired by this, we design a task-specific learning curriculum and construct a self-paced curriculum learning based model specifically for our task.

Network Pruning: DeepIoT [54] compresses neural network structures into smaller dense matrices byfi nding the minimum number of non-redundant hidden elements while maintaining application-level accuracy. In [23], authors suggested removing weights below a threshold, followed byfi ne-tuning to recover its accuracy. However, the model loses universality andfl exibility [34]. There are some data-driven approaches, e.g., [21, 34], where authors use randomly sampled feature maps or data points in pruning networks which may make model biased towards sampled data-points. Group-sparsity regularization with loss function has been used in [30] to regularize the structure of the network and to create a sparse network. $l1$ norm based pruning was introduced in [6, 23], which generates a sparse model. However, specialized software is required for efficient inference from these sparse models and most of these models are not perfectly supported by off-the-shelf deep learning libraries. In contrast to these approaches, we propose an $l_{2,1}$ norm based representativefi lter selection and pruning method. Our approach does not require any data at the time of pruning and also retains the original model structure. Our approach is perfectly supported by any off-the-shelf libraries.

Other Solutions: FORK [37, 38] uses a depth sensor mounted at a ceiling to detect humans on an ARM processor. It detects heads and shoulders by leveraging the anthropometric properties of human bodies. In [36], authors propose to mount an 8x8 pixel IR array sensor at the side of a door to detect and count people. However, it just uses temperature difference compared to the background to detect humans instead of detecting any specific body parts. Hence, any warm objects will be detected as humans. In [27], the authors use two Unmanned Aerial Vehicles (UAVs) to perform 3D through-wall imaging using WiFi RSSI. The solution uses Markov random field modeling and can detect simple objects, e.g., L-shaped structures. Sonicdoor [28] proposes a technique to detect and identify around 100 occupants by sensing their body shape, movement, and walking patterns as they walk through a door instrumented with three ultrasonic sensors. In [26], authors mount a radar sensor at a doorway and use the reflected energy coming from different body parts to create a weak biometric profile for differentiating people. In [5], authors use radio tomography to detect and localize people outdoors. In [57], authors propose to use a combination of thermal sensors and ultra-low power coarse stereo cameras to detect the presence of body parts at wearable power budgets. In [40], authors propose to track multiple people in a dynamic industrial environment by leveraging CCTV camera along with the radio and inertial sensors of each worker's mobile phone. In [39], authors utilize footstep induced structural vibration to identify occupants. In [53], authors count and localize multiple individuals using radio signal strength. In [7], authors present a radio-optical based recognition system where a radio-optical transmitter emits a beacon whose IR signal strength is used to track the orientation of tagged objects.

3 OVERVIEW

We use the depth sensor in Kinect for XBOX One [1] (Figure 2(a)) in this work. We mount the depth sensor at the ceiling looking downwards near a doorway as shown in Figure 1. It has a 0.5 - 8 meters range and 70°x60° field-of-view (FOV). The range is good enough to detect objects through doorways as the average height of afl oor is 10 feet [3]. For doing the computation, we use an Nvidia Jetson TX1 development kit [2] that contains a TX1 embedded GPU under the heat sink (Figure 2(b)). It has 256 CUDA cores with 4 GB memory. It can perform 512 GFLOPS FP32 operations. To

(a) (b)

Figure 2: (a) Kinect for XBOX One. (b) Nvidia Jetson TX1 Development Kit.

compare with a more powerful GPU, NVidia K40 has 2880 CUDA cores with 12 GB memory that can perform 4291-5040 GFLOPS FP32 operations.

ODDS is implemented in C++. It runs on Ubuntu 16.04 as an application on Jetson TX1 development kit. It uses *libfreenect2* library of the OpenKinect project to get the depth frames from a Kinect, which in turn uses *libusb* library to access the USB interface. It uses Caffe [24] framework to run our CNN model to detect objects. However, other CNN frameworks, e.g., TensorFlow, Torch could also be used. Note that it performs the entire processing on board and publishes only the timestamp and object metadata to a server. However, authorized users can configure ODDS to publish images to the server for a specific type of object, e.g., a gun. Authorized users can monitor the reported activities using a browser in real-time. An overview of the ODDS object detection approach is shown in Figure 3, where for an input depth frame, ODDS performs a single evaluation of the CNN model and produces bounding box offsets for all detected objects, along with the class of each object and the corresponding confidence score. Our detection CNN uses feature map from six selected layers to detect objects of different scales. These six layers are shown inside the detection CNN in Figure 3. ODDS uses different modules to perform different tasks tofi nally output the object detection and classification information in real-time. The ODDS software architecture and the modules are shown in Figure 4. The raw depth frames are pre-processed to be ready for input to CNN in Depth Preprocessor module. Object Detection CNN module uses the trained CNN to both detect and classify objects from depth frames in real-time. The Detection Post-processor module is responsible for removing detections with low confidence and merging multiple detections that belong to the same object. The Update Reporter module reports the detection output to a server. In the next section, we describe how we build the CNN model that is lightweight enough to be able to run on this embedded platform and also accurate.

4 METHODOLOGY

We perform an empirical study of directly using existing pre-trained CNN models that are trained using RGB images to see how they perform on our data. Wefi nd that the performance is extremely poor (accuracy less than 1%) and they are hardly able to classify or detect any object. We believe this is because of significant difference in RGB and raw noisy depth images. Moreover, no object detection dataset of depth images is available for our setting, where the sensor is mounted at the ceiling looking downwards. Hence, we collect data and build our own dataset [4]. Inspired by the success of CNN based single shot object detectors, we construct an object detection CNN and train it utilizing our dataset. Our depth dataset has limited variety and data imbalance between classes. To help the network reach its potential in spite of these constraints, we develop a training strategy which helps the model to achieve better performance. To help the network achieve higher accuracy, we design afi xed path curriculum learning strategy based on the complexity of training images and use that to guide the training process. Inspired by the success of sparse coding in representative subset selection and regularizing deep networks [16, 35, 58], we propose a sparse coding based approach to improve the speed of detection process, which works on retaining most representativefi lers and prune redundant ones. We also perform extensive data augmentation, which is a common and good practice in deep network training with limited data.

In this section, wefi rst present our depth dataset construction and annotation process (Section 4.1). Then, we present our deep CNN based approach for single-shot object detection (Section 4.2), followed by our proposed training process (Section 4.3). The curriculum learning based strategy to train our object detection CNN is described in Section 4.3.2. Then we describe how we use sparse coding based representative selection to improve the speed of detection process in Section 4.3.3.

4.1 Smart Building Depth Dataset Construction

As discussed in Section 1 and 2, the best solutions for object detection are based on deep CNN models that require millions of RGB images to train [29, 50]. To the best of our knowledge, there is no depth dataset for our proposed mounting position. Acquisition of such a depth dataset is more challenging than that of an RGB dataset. Because, RGB images can be crawled from the internet using crowd-workers, whereas we need a specialized set up to collect depth data. Also, we need annotation for bounding boxes of each object, which is labor-intensive. In this section, we describe how we collect our depth dataset and how we annotate depth frames in an efficient way using multiple instance learning (MIL) based tracking algorithm.

Data Collection: We collected data from two settings. We had a Kinect mounted at a 9.3 feet ceiling near to a 6 feet wide door. We also used a tripod with a horizontal extender holding the Kinect at a similar height looking downwards. We asked 20 volunteers to enter and exit 9 times each in different directions (3 times walking straight through the center, 3 times walking towards the left side, 3 times walking towards the right side) holding objects in many different ways and poses underneath the Kinect. Each subject was using his/her own backpack, purse, laptop, etc. As a result, we considered varieties within the same object, e.g., for laptops, we considered Macbooks, HP laptops, Lenovo laptops of different years and models, and for backpacks, we considered backpacks, side bags, and purse of women. We asked the subjects to walk while holding it in many ways, e.g., for laptop, the laptop was fully open, partially closed, and fully closed while carried. Also, people hold laptops in front and side of their bodies, and underneath their elbow. The subjects carried their backpacks in their back, in their side at different levels

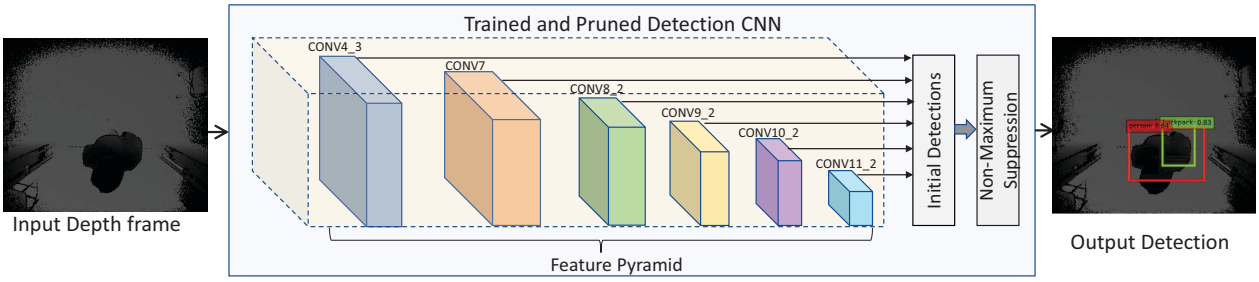

Figure 3: An overview of how ODDS works

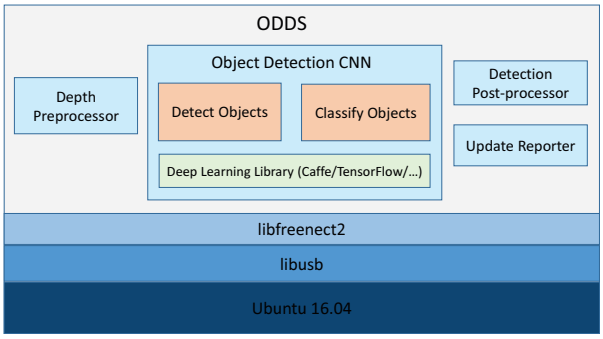

Figure 4: ODDS Software Architecture

from foot to shoulder. We wanted to collect data with real guns. However, bringing real guns to the office is prohibited. So, we obtained a few nerf guns and the subjects were carrying these guns pointing it to front, side, up, and down while walking. Figure 5 shows a few samples of our collected data containing objects from backpack class (row 1, 2) and laptop class (row 3, 4). It is really hard to see these objects from the depth frames. So, we also show the corresponding RGB images in Figure 5. The frame rate for data collection was around 15 FPS.

Annotation using MIL tracking with human in the loop: Our data labeling system hinges on online multiple instance learning (MIL) based tracking [8] with human (annotator) in the loop. The MIL algorithm trains a classifier in an online manner to separate the object from the background. MIL is one of the most stable trackers and is capable of handling ambiguously labeled samples that are provided to the tracker itself. The MIL based approach extracts a bag of potentially positive image patches and has the flexibility to pick out the best one. Additionally, the algorithm is simple to implement and can run at real-time.

For all sequences, the annotator will start the process by drawing the bounding box of the object in thefi rst frame. Then, the tracker starts tracking the object frame by frame (while the subject is entering/exiting through the door) and the bounding box coordinates are stored. When the annotator feels that the tracker has drifted significantly from actual track, he stops the tracker, draws the bounding box around the object for that frame and starts the tracking again. This process continues till the end of the sequence (the subject entered/exited). Using MIL in labeling decreases the tracking time significantly but it may introduce slight noise in the annotation. We believe that small noise in annotation ultimately will help our CNN model to generalize better. The dataset statistics are shown in Table 1. There was a person in 65084 frames. There was a backpack, laptop, gun in 21948, 20693, and 19379 frames, respectively. Only 558, 670, 777, and 1059 frames contain a phone, an umbrella, a cup, and a box, respectively. It is obvious that our dataset is highly imbalanced.

4.2 Object Detection Network

Our network structure is inspired by SSD object detection network [32]. We evaluate popular state-of-the art object detectors on our task, e.g. Faster RCNN [43], YOLO [42], and SSD [32] by training on our dataset and empirically select SSD for better performance in terms of speed and accuracy. Faster RCNN and SSD both achieve good accuracy, but Faster RCNN is computationally more expensive. YOLO is computationally fast, but the accuracy is relatively low.

Network Overview: The object detection network is built on top of a base pre-trained CNN (e.g. atrous VGG-16 [13] in our case). Series of progressively smaller convolutional layers are added to the base network that have feature maps of progressively decreasing spatial resolution. Different layers within a CNN are used to predict objects of different scales. We use 6 feature maps (see Figure 3) with different scales (e.g. 16 x 16, 8 x 8, 4 x 4 etc), each responsible for a different scale of objects, allowing it to identify objects across a large range of scales. Feature map with highest spatial resolution is responsible for detecting smallest objects and vice versa. A small set of default boxes of different aspect ratio (e.g., $1, 2, 3, \frac{1}{2}, \frac{1}{3}$) is evaluated using a 3x3 convolutionalfi lter at each location in each of these feature maps. At each location, 4 shape offsets and the confidences for all object categories are simultaneously predicted for each default box. Hence, for each input depth image, the network ultimately produces a constant number of bounding boxes and classification scores of all categories for each box. Combining predictions from all locations of feature maps for the default boxes with several scales and aspect ratios allows the model to predict objects of different sizes and shapes with high accuracy. Finally, these detections are post-processed by non-maximum suppression to produce thefi nal detection results. Non-maximum suppression groups together overlapping boxes and only the box with highest confidence score is kept from each group. For more details on the network architecture, we refer the interested readers to [32].

Figure 5: A few samples of collected images of 'Backpack' and 'Laptop' classes from our dataset.

Class Name	Person	Backpack	Box	Cup	Gun	Laptop	Phone	Umbrella
Total No. of Images	65084	21948	1059	777	19379	20693	558	670

Table 1: Number of depth frames per class in our dataset.

The training objective is to minimize a weighted sum of localization loss and classification confidence loss. The objective loss function is:

$$Loss(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + L_{loc}(x, l, g)) \quad (1)$$

Here, N is the total number of default boxes matched to a ground truth box, x denotes training data, c denotes confidences of multiple classes, l denotes predicted box parameters, and g denotes the ground truth box parameters.

The localization loss is a smooth L1 regression loss between the predicted bounding box parameters and the ground truth bounding box parameters. The class confidence loss is softmax loss over confidence of multiple classes. To compute the loss, SSD matches objects with default boxes of different aspects with ground truth boxes based on intersection over union (IoU). For each ground truth box, it is matched with the default boxes having Jaccard overlap higher than a threshold (e.g. 0.5). This allows one ground truth box to be matched to multiple default boxes rather than selecting only one default box with the highest overlap [32].

4.3 Training ODDS for Object Detection

In the previous section, we have presented a supervised object detection approach for detecting bounding boxes and object classes from an image. However, to achieve optimal performance, a few practical concerns have to be taken care of, e.g., the imbalanced number of images in different classes, the complexity of training examples in different classes and the limited number of training examples in the dataset. To handle the issue of imbalance in object classes and complexity, we introduce a specialized curriculum based training strategy that follows the natural complexity and imbalance found in our dataset. In addition, we perform extensive data augmentation in training for this purpose. In order to speed up, we develop a sparse coding based model pruning method for our object detection module. There are mainly two ways to reduce the detection time of an object detection module. First, we can start with a smaller/lighter pre-trained CNN network (e.g., AlexNet [29], ZFNet [56], ThiNet [34]) as the base network, instead of using VGG-16 [46]. Second, we can train the object detection network with a large pre-trained network, which is capable of learning complex models suitable for achieving high accuracy for our case and then design a deep CNN reduction method to accelerate the detection process. We focus on the second approach as it provides a more accurate model. We initially train the object detection network as described in Sec. 4.3.2 and prune this trained model with our proposed mixed $l_{2,1}$ norm based sparse representation approach, as described in Sec. 4.3.3.

4.3.1 Preprocessing. The resolution of a depth frame from a Kinect for XBOX One is 512x424. Each pixel of the depth frame provides the distance from the Kinect to the nearest object in millimeters. The maximum range of Kinect for Xbox One is 8 meters, which means that each depth pixel value can be represented by a

13 bit unsigned integer (2^{13} > 8000). We divide each depth pixel value by 32 (i.e., 2^5), which converts each depth pixel value to an 8 bit integer. There are two advantages of this divide:

(1) It helps us discarding minor noise in the depth estimation, which helps in training the model.
(2) The transformed frame can be treated as a single channel RGB frame. We replicate the same frame 3 times over 3 channels to form an RGB image using pure depth data. Converting it to an RGB frame enables us to reuse weights from pre-trained models for transfer learning.

Even though we replicate the same depth channel thrice to make fake RGB frames, this replication has a minimal impact on the computation time and memory footprint of a sufficiently deep CNN like ours. For the ODDS (no pruning) network shown in Table 2 (22.935 million number of parameters and 96.43 MB model size), if a one channel depth image is used as input to a similar CNN model, the number of parameters decreases by only 1152 and the model size decreases approximately by 4.5 KB. Hence, these redundant three channels are increasing the model size by only 0.0046%. A good initialization is crucial for a deep CNN to converge and converting depth frames to fake RGB frames enables us to utilize weights from pre-trained RGB models. The reason for reusing weights from existing pre-trained RGB models is because RGB images based object detection literature is much more richer than the depth sensing based counterpart.

We also use several data augmentation techniques to reduce the over-fitting problem because of limited data and to make our model more robust to different noise, sizes and, shapes. Following [22, 32], we randomly sample several patches from images and apply different affine transformations and photo-metric transformations on the images. For more details on data augmentation, we refer the interested readers to [22, 32].

The resolution of the input image to the network is a design choice. Using a very high resolution image as an input is expected to provide a better accuracy, but the processing time may be significantly higher. On the other hand, a lower resolution image may have a higher speed, but the accuracy may suffer. We empirically choose $300x300$ resolution as the input image size of our network. Hence, we re-size the $512x424$ depth images to $300x300$ before using them as input to our network.

4.3.2 Curriculum Learning . In deep neural networks, the model parameters are learned in an iterative fashion using stochastic gradient descent and its variations. As the objective function in deep networks has a highly non-convex shape, the order of sample presentation for such networks is important. It has been shown that appropriate curriculum strategies guide the learner towards better local minima [11]. Curriculum learning allows the model to learn simpler instances (e.g., cleaner examples, simpler shapes)first so they can be used as building blocks to learn more complex ones, which leads to a better performance in thefi nal task.

In our work, we propose to extend the concept of curriculum learning for object detection. Utilizing the prior knowledge of easy and hard examples in our dataset, we design a curriculum learning strategy for our task, which improves the model performance significantly. Wefi nd that person class is the most complex one in our training set. The person class has more varied shapes than other classes and there is a high chance of having noisy bounding box annotation in the person class because of overlap with other classes. Moreover, the person class has significantly more images than other classes, as we are interested in the objects carried by humans in the office and all of our training images have at least one person in it.

Based on this, we design afi xed path curriculum for our task. In the training procedure, the complexity of the training data should be increased while the simpler image samples should not be discarded. Wefi rst perform training on a subset (D_1), which contains all the classes except the person class, until convergence. Then, the hard subset (D_2), i.e., person class is merged into the dataset. The best model weights W^* from the previous training step are adopted as the initialization for next training iteration. Then, the training is performed again with a lower basic learning rate to reduce the influence of difficult samples. CNN with curriculum learning algorithm is shown in Algorithm 1.

Algorithm 1 Training CNN with Curriculum Learning

Input: Input Dataset $D = \{D_1, D_2\}$, where, D_1 contains all samples without Person class and D_2 contains only samples of Person class.
i=1
$D_{train} = \Phi$
Initialize network parameters W using a pre-trained CNN and Xavier algorithm
while (i<=2) **do**
  $D_{train} = D_{train} \cup D_i$
  while (not converged) **do**
    Perform training, $train(D_{train}, W)$
  end while
  Select best W^*;
  Update learning Rate;
  $i = i + 1$
end while
Output: Optimal Model Parameter W^*

4.3.3 Model Pruning. After our model is trained, we perform sparse coding based channel pruning andfi ne-tuning (retraining using Algorithm 1) so that ODDS can run on a low cost embedded platform in real-time. The goal of model pruning is tofi nd a smaller representativefi lter set from thefi lters of a layer. In particular, we are trying to represent thefi lters of a layer by selecting only a few representativefi lters. Our representativefi lter selection is based on sparse representative selection approach [14, 16]. The basic idea behind this is to utilize the self-expressiveness property, which states that each point in the set can be described as a linear combination of a few of the selected representative points.

Let, $X \in \mathbb{R}^{B \times N}$ be the matrix representing all thefi lters in a layer, where $X = \{x_i \in \mathbb{R}^B, i = 1, \cdots, N\}$. Each x_i represents the components of afl attenedfi lter in the current layer, which is B-dimensional. N denotes the number offi lters in the layer. For example, in VGG-16, conv1_1 layer has 64fi lters of dimension $3x3x3$. Wefl atten thefi lter to make it 1-dimensional for easier calculation. Afterfl attening, the size offi lter matrix of conv1_1 is

$27x64$. Now, the natural goal is to establish afi lter level sparsity which can be induced by performing l_1 regularization on rows of the selection matrix $Z \in \mathbb{R}^{N \times N}$. By introducing the row sparsity regularizer, the problem can now be succinctly formulated as

$$\min \ ||X - XZ||_F^2 \text{ s.t. } ||Z||_{2,0} \leq \tau, \tag{2}$$

where, $\|Z\|_{2,0}$ gives the number of nonzero rows of the matrix Z. τ is a tradeoff parameter. Solving Eq (2) is an NP-hard problem since it requires searching over every subset of the τ columns of X. A standard relaxation to the constraint 2 is given by

$$\min \ ||X - XZ||_F^2 \text{ s.t. } ||Z||_{2,1} \leq \tau, \tag{3}$$

where, $Z \in \mathbb{R}^{N \times N}$ is the sparse coefficient matrix and $\|Z\|_{2,1} \triangleq \sum_{i=1}^{N} \|z_i\|_2$ is the row sparsity regularizer, i.e., sum of l_2 norms of the rows of Z.

Minimization of Eq. 3 leads to a sparse solution for Z in terms of rows, *i.e.*, the sparse coefficient matrix Z contains a few nonzero rows that constitute the representative set. Optimization of Eq. 3 attempts to obtain a sparse set of representativefilters.

Optimization: Here, we briefly describe the strategy to solve the convex optimization problem in Eq. 3. We solve the optimization using an Alternating Direction Method of Multipliers (ADMM) framework [12]. Using Lagrange multipliers, optimization problem in Eq. 3 can be written as

$$\min \frac{1}{2}\|X - XZ\|_F^2 + \lambda \|Z\|_{2,1} \tag{4}$$

where λ is trade-off parameters associated with sparsity and diversity regularization. Using ADMM with an auxiliary variable A, Eq. 4 can be equivalently expressed as

$$\min \frac{1}{2}\|X - XA\|_F^2 + \lambda \|Z\|_{2,1} \ s.t. \ A = Z, \tag{5}$$

Now, we can form the augmented Lagrangian $L_\rho(A, Z, \Lambda)$ with both linear and quadratic terms as following,

$$L_\rho(A, Z, \Lambda) = \frac{1}{2}\|X - XA\|_F^2 + \lambda \|Z\|_{2,1} + \Lambda^T(A - Z) + \frac{\rho}{2}\|A - Z\|_F^2 \tag{6}$$

The above Lagrangian is a function of two primal(Z, A) and one dual(Λ) variable. Thus, ADMM consists of minimization of Eq. 6 alternatively with respect to these three variables.

After performing the derivatives with respect to each variable and equating to zero, wefi nd the ADMM consists of following iterations,

$$\begin{aligned} A^{(t+1)} &:= (X^T X + \rho I)^{-1}(X^T X + \rho(Z^{(t)} - \Lambda^{(t)}/\rho)) \\ Z^{(t+1)} &:= S_{\lambda/\rho}(A^{(t+1)} + \Lambda^{(t)}/\rho) \\ \Lambda^{(t+1)} &:= \Lambda^{(t)} + \rho(A^{(t+1)} - Z^{(t+1)}) \end{aligned} \tag{7}$$

In Eq. 7, the minimization respect to $l_{2,1}$ norm is defined as,

$$S_\mu(z) = \max\left\{||z||_2 - \mu, 0\right\}\frac{z}{||z||_2}. \tag{8}$$

We choose columns of X corresponding to the nonzero rows of final Z as the representativefi lter and denote the matrix of the representativefi lters as Y. Here, $Y \in \mathbb{R}^{B \times K}$ is the feature matrix for all selected representativefi lters, where $Y = \{y_i \in \mathbb{R}^B, i = 1, \cdots, K\}$. y_i represents the components of ith representativefi lter which is B-dimensional. K denotes the number offi lters in representative set. After pruning is done, wefi ne-tune the pruned model to regain the performance.

4.3.4 Model Training Details. Our network architecture consists of a pre-trained convolutional neural network followed by smaller convolutional layers. We use ReLU nonlinear activations throughout the network. The convolutional network module follows the VGG-16 architecture [46]. We trained the CNN using stochastic gradient descent. We start thefi rst step of training with 0.0005 learning rate and decrease it when the training loss reaches a plateau. For the second step of training, we start with a learning rate of 0.00025. We use 0.9 momentum, 0.0005 weight decay, and mini-batch size 32. The new layer weights are initialized using the Xavier algorithm [18]. The entire network isfi ne-tuned end-to-end for the task. We follow an 80% / 20% split for training image set vs. testing set. We prepare our depth dataset structure similar to the pascal-VOC format [17], which helps in handling learning procedure easily with different existing object detection networks.

5 EVALUATION

In this Section, we evaluate the performance of ODDS for precision and speed. We randomly split the dataset for training (80%) and testing (20%) and evaluate performance on the testing set. Note that this 20% split is done per class. Hence, for classes with small samples, e.g., a cup or a phone, we still have 20% images of that class in the test set.

We train several different models tofi nd an optimal network in terms of accuracy and speed. The networks vary based on 3 factors: (1) base pre-trained CNN used (ZFNet, ThiNet, or VGG16), (2) use of training strategy (curriculum learning is used or not), and (3) use of pruning step (sparse coding basedfi lter selection is used or not). We try two different pruning network, i.e., low pruning and high pruning. The detailed statistics of the number offi lters and parameters in the ODDS with no pruning, ODDS with low pruning, and ODDS with high pruning model is shown in Table 2 for VGG16 based models. We only show the convolutional layers, where the number offi lters are pruned and the number of parameters are decreased.

The mAP (mean Average Precision) of different methods is shown in Table 3. The base network and technique used for training can be identified based on the name of the network. For example, ODDS-VGG16 is trained using the atrous VGG16 [13] as the base network and a single training step. Pruning step and curriculum based training strategy is not used. On the other hand, ODDS-VGG16+CurrLearning+Pruning (Low) is trained using VGG16 as the base network and proposed two step curriculum based training (Sec. 4.3.2) and the sparse coding based low pruning step is also used. We also show performance when high pruning is applied. We use the mAP criterion followed in the PASCAL VOC object detection challenge [17], which is a standard metric for evaluating the performance of object detectors.

Table 3 shows that curriculum based training guides ODDS to achieve better accuracy. ODDS with curriculum based training achieves the highest mAP of 84.94%. The performance increases significantly for small objects with curriculum based training (e.g., 68.42% from 35.89% for phones, 71.57% from 63.32% for cups), compared to training without it, whereas for large objects the mAP

Network / Layers	ODDS (No Pruning)		ODDS (Low Pruning)		ODDS (High Pruning)	
	No. of Filters	No. of Weights	No. of Filters	No. of Weights	No. of Filters	No. of Weights
Conv1_1 (3X3)	64	(3*3*3)*64 = 1,728	32	(3*3*3)*32= 864	16	(3*3*3)*16 = 432
Conv1_2 (3x3)	64	(3*3*64)*64 = 36,864	32	(3*3*32)*32 = 9,216	16	(3*3*16)*16 = 2,304
Conv2_1 (3x3)	128	(3*3*64)*128 = 73,728	64	(3*3*32)*64 = 18,432	32	(3*3*16)*32 = 4,608
Conv2_2 (3x3)	128	(3*3*128)*128 = 147,456	64	(3*3*64)*64 = 36,864	32	(3*3*32)*32 = 9,216
Conv3_1 (3x3)	256	(3*3*128)*256 = 294,912	128	(3*3*64)*128= 73,728	64	(3*3*32)*64 = 18,432
Conv3_2 (3x3)	256	(3*3*256)*256 = 589,824	128	(3*3*128)*128 = 147,456	64	(3*3*64)*64 = 36,864
Conv3_3 (3x3)	256	(3*3*256)*256 = 589,824	128	(3*3*128)*128 = 147,456	64	(3*3*64)*64 = 36,864
Conv4_1 (3x3)	512	(3*3*256)*512 = 1,179,648	256	(3*3*128)*256 = 294,912	128	(3*3*64)*128 = 73,728
Conv4_2 (3x3)	512	(3*3*512)*512 = 2,359,296	256	(3*3*256)*256 = 589,824	128	(3*3*128)*128 = 147,456
Conv4_3 (3x3)	512	(3*3*512)*512 = 2,359,296	512	(3*3*256)*512 = 1,179,648	512	(3*3*128)*512 = 589,824
Conv5_1 (3x3)	512	(3*3*512)*512 = 2,359,296	512	(3*3*512)*512 = 2,359,296	128	(3*3*512)*128 = 589,824
Conv5_2 (3x3)	512	(3*3*512)*512 = 2,359,296	512	(3*3*512)*512 = 2,359,296	128	(3*3*128)*128 = 147,456
Conv5_3 (3x3)	512	(3*3*512)*512 = 2,359,296	512	(3*3*512)*512 = 2,359,296	128	(3*3*128)*128 = 147,456
fConv6 (3x3)	1024	(3*3*512)*1024 = 4,718,592	1024	(3*3*512)*1024 = 4,718,592	512	(3*3*128)*512 = 589,824
fConv7 (1X1)	1024	(1*1*1024)*1024 = 1,048,576	1024	(1*1*1024)*1024 = 1,048,576	1024	(1*1*512)*1024 = 524,288
Conv6_1 (1x1)	256	(1*1*1024)*256 = 262,144	256	(1*1*1024)*256 = 262,144	128	(1*1*1024)*128 = 131,072
Conv6_2 (3x3)	512	(3*3*256)*512 = 1,179,648	512	(3*3*256)*512 = 1,179,648	512	(3*3*256)*512 = 1,179,648
Conv7_1 (1x1)	128	(1*1*512)*128 = 65,536	128	(1*1*512)*128 = 65,536	64	(1*1*512)*64 = 65,536
Conv7_2 (3x3)	256	(3*3*128)*256 = 294,912	256	(3*3*128)*256 = 294,912	256	(3*3*64)*256 = 147,456
Conv8_1 (1x1)	128	(1*1*256)*128 = 32,768	128	(1*1*256)*128 = 32,768	64	(1*1*256)*64 = 16,384
Conv8_2 (3x3)	256	(3*3*128)*256 = 294,912	256	(3*3*128)*256 = 294,912	256	(3*3*64)*256 = 147,456
Conv9_1 (1x1)	128	(1*1*256)*128 = 32,768	128	(1*1*256)*128 = 32,768	64	(1*1*256)*64 = 16,384
Conv9_2 (3x3)	256	(3*3*128)*256 = 294,912	256	(3*3*128)*256 = 294,912	256	(3*3*64)*256 = 147,456
Total Parameters	-	22,935,232 = 22.935 Million	-	17,801,056 = 17.801 Million	-	4,769,968 = 4.769 Million
Model Size		96.43 MB		76.36 MB		23.02 MB

Table 2: Number of parameters in different layers of ODDS-VGG16 network with and without pruning

Methods	**mAP**	**Laptop**	**Backpack**	**Umbrella**	**Phone**	**Box**	**Cup**	**Gun**	**Person**
ODDS-ZFNet	55.49	76.77	83.53	30.56	0.65	87.88	20.21	60.72	82.25
ODDS-ThiNet	49.88	82.52	86.33	17.08	1.13	48.95	15.93	84.63	63.35
ODDS-VGG16	78.94	**91.5**	**94.53**	77.37	35.89	**90.05**	63.32	90.49	89.34
ODDS-VGG16+CurrLearning	**84.94**	90.59	90.83	**90.66**	**68.42**	80.86	**71.57**	**95.96**	**90.68**
ODDS-VGG16+ Pruning (Low)	75.43	90.08	90.76	84.17	18.12	88.12	54.24	90.79	87.12
ODDS-VGG16 + Pruning (High)	51.59	78.79	85.44	13.83	3.15	43.04	19.11	80.25	89.15
ODDS-VGG16 + CurrLearning + Pruning (Low)	80.14	90.52	90.84	82.31	46.67	83.68	66.50	90.89	89.78
ODDS-VGG16 + CurrLearning + Pruning (High)	71.35	85.53	90.20	78.28	21.12	72.40	46.26	88.85	88.18

Table 3: Object detection results in mAP. The best score is indicated as bold and the second best score is underlined

remains almost the same. It is also worth indicating that when using a smaller base network initially, the mAP is low, e.g., 49.88% for ODDS-ThiNet. ThiNet is a reduced VGG model for better speed. However, our pruned model, which we set to achieve same network structure of ThiNet achieves significantly higher mAP with higher frames per second. After applying pruning, the mAP reduces to 80.14%, which is close to the highest mAP. However, it speeds up ~2x as described later. If we keep pruning the network, the performance drops significantly as we see the mAP drops to 71.35% after applying high pruning.

The computation time of complete object detection process is shown in Table 4 for thefi ve best performing models of Table 3. We run ODDS on two powerful GPUs: GTX 1080 Ti and K40, and on two embedded GPUs: Jetson TX1 and TK1. We see that FPS gets almost doubled after applying low pruning across all GPUs (69.33

Method	mAP	Model Size (MB)	Frame Per Seconds in GPUs			
			GTX 1080 Ti	K40	Jetson TX1	Jetson TK1
ODDS-VGG16	78.94	96.43	41.5	14.85	3.25	1.32
ODDS-VGG16+ CurrLearning	84.94	96.43	41.5	14.85	3.25	1.32
ODDS-VGG16+ Pruning (Low)	75.43	76.36	69.33	26.1	5.94	2.47
ODDS-VGG16 + CurrLearning + Pruning (Low)	80.14	76.36	69.33	26.1	5.94	2.47
ODDS-VGG16 + CurrLearning + Pruning (High)	71.35	23.02	104	42.6	11.93	3.79

Table 4: Speed, Accuracy, Model-Size trade-off chart for the 5 best performing models of Table 3

from 41.5 in GTX 1080 Ti, 26.1 from 14.85 in K40, 5.94 from 3.25 in TX1, 2.47 from 1.32 in TK1), which means a 1.75-2x speed-up in execution time by keeping the mAP close. This is because the pruned model has 79% parameters (76.36 MB vs. 96.43 MB). After applying high pruning, we get a 2.5-4x speed up from the original ODDS-VGG16+CurrLearning model. This is because the high pruning model has only 23% parameters (23.02 MB vs 96.43 MB). However, the mAP drops to 71.35%. Note that 3 FPS is reasonable enough to track people for such a setting as shown in [37]. Hence, Jetson TX1 can be used with ODDS-VGG16+CurrLearning+Pruning (Low) model that can detect objects with 80.14% mAP at 5.94 FPS. If the application requires a cheaper solution where lower accuracy is permitted, ODDS-VGG16+CurrLearning+Pruning (High) model can be used on a Jetson TK1 with 71.35% mAP at 3.79 FPS. We could not perform a real-time evaluation of ODDS as the speed of ODDS starts to degrade significantly if we start saving depth frames to a disk due to I/O. However, for that evaluation, we need to save depth frames to compare the bounding boxes of the detected objects with that of ground truth. By default, ODDS does not store any depth frames for privacy concerns. However, Table 4 provides an insight regarding the real-time performance of ODDS as it shows the number of frames processed per second by ODDS.

Figure 6 shows a few frames demonstrating objects and associated bounding boxes detected by ODDS. Note that ODDS can detect multiple people and objects in a frame, even though none of the training samples had multiple objects in it. Initially, we experienced an issue with detecting multiple humans. ODDS was detecting multiple objects, e.g., multiple backpacks, multiple laptops, multiple backpacks and laptops in a frame. But, it was not detecting multiple people in a frame. By going through the training set, we realized that there were ~400 frames where there were multiple people, but only one human was annotated. As a result, ODDS assumed the rest as background. Removing these frames and retraining of ODDS enabled the detection of multiple people simultaneously. Also, the solution generalizes to other doors. We run ODDS in two different doors, where no data have been collected during dataset construction and it seems to perform accurately (see Figure 6).

6 DISCUSSIONS

Many cyber-physical systems use various types of sensors for detecting an event of interest, e.g., using body-worn sensors for activity recognition or Wifi imaging for detecting objects/locations. Usually, the event detection is performed using signal processing techniques. An object detector can be a powerful solution for such problems. Because, these sensor data can be converted to an image, where X-axis will contain time and Y-axis will contain multiple sensing modalities. Given enough training data for each event of interest, an object detector like our proposed solution will determine the corresponding pattern (object) of each event of interest. As an example, using this technique, we might detect the start time and end time of eating activity using body-worn sensors, as an object detector will be able to locate the exact location of these two objects in an image (containing temporal data). The challenges will be how to decide the resolution of the image, how to collect clean data and accurately annotate these so that background data (non-eating activities) are different than events of interest (eating activities).

ODDS takes several measures to address privacy concerns of the occupants. First, it performs the entire processing onboard and it does not store or upload images. Second, the depth sensor is mounted at the ceiling looking downwards. So, ODDS does not see the face of the subject unless s/he looks at the ceiling. Third, it uses a depth sensor instead of using an RGB camera, which is a lot less privacy invasive than an RGB camera for the following reasons: (a) A depth sensor does not reveal the color of the clothes, skin, and hair. It is very hard to determine clothing level by looking at a depth frame. (b) RGB (even grayscale) cameras expose privacy risks if they are compromised when connected to the Internet, e.g., if a whiteboard is in thefi eld of view of the camera, some invention ideas or a prototype product (with labels) from an office could be revealed outside. ODDS is not as privacy-invasive even if compromised as the sensor will report afl at surface for the whiteboard. (c) As our depth sensor only sees the body shape and many people have similar body shape, it cannot identify someone where many people arrive, e.g., in a shopping center or in an academic building. It is debatable whether using a depth sensor makes more sense than using an RGB camera for security applications. If there is an intruder with a gun, it may be helpful to capture an RGB image and spread it to the people of interest. Someone can argue that it is allowed to carry guns in some places and hence it is privacy invasive of the gun bearer to take photos of him and spread it around. However, regardless of the law of the land, if we can notify people that one of them is carrying a gun, that could help them in taking precautionary steps. While an RGB camera can help identifying the carrier, it can give false positives if someone is wearing a T-shirt with a photo of a gun. Even though it makes sense to have RGB cameras at the main entrances of a building, deploying RGB cameras throughout a building can be perceived to be very privacy invasive, especially now-a-days when security breaches are commonplace. Also, it is hard to convince occupants to deploy

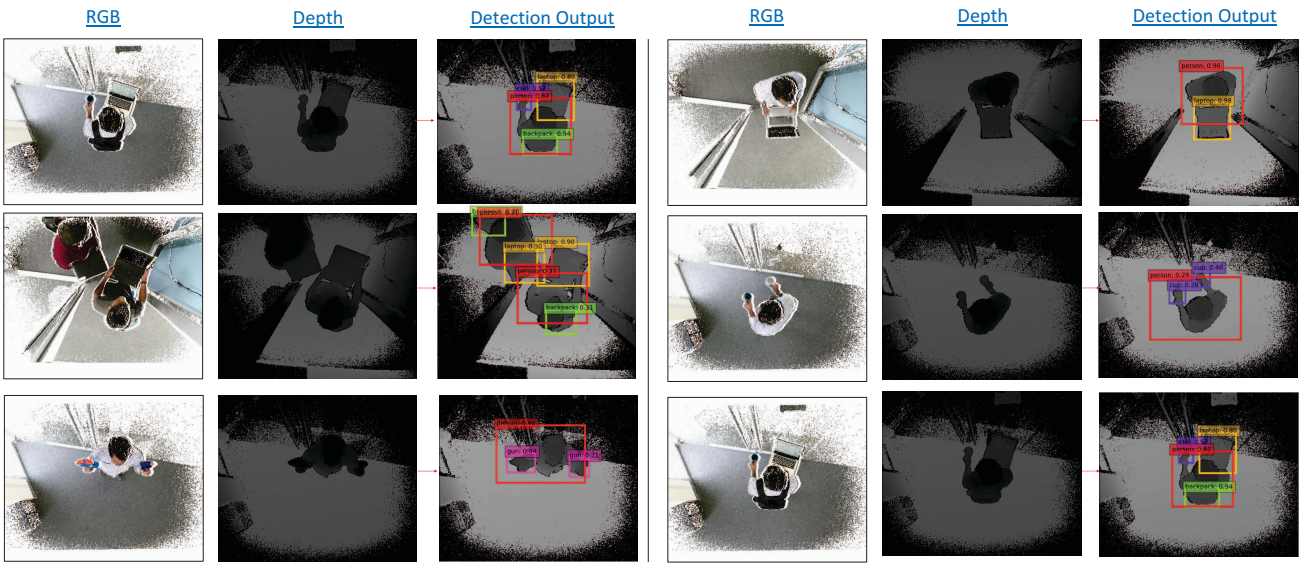

Figure 6: Example of randomly selected 6 depth frames and corresponding detection output. The paired RGB frames with depth noise are shown to the left to give a better understanding of the object in the image and noise in raw depth frames. Note that no data have been collected from these two doors during dataset construction.

RGB cameras in their offices for improving energy efficiency or for a threat detection application. A depth sensor can help in these cases.

There are also several advantages of using only depth data instead of using an RGBD sensor. First, it is less privacy invasive than an RGB camera as mentioned above. Second, it is less expensive than an RGBD sensor as it does not need the corresponding RGB sensor. Third, it requires less computation at the inference phase while running CNNs as it does not use the 3 channels of RGB cameras. Also, it saves a lot of computation at the driver level as the resolution and frame rate of an RGB camera is usually different than that of the depth sensor and hence a fair amount of mapping, translation, and synchronization happens between frames (RGB, D) in the driver that we avoid completely. Note that a depth sensor does not see the color of an object, which means that our solution will not be able to classify two boxes of same size but of different colors. However, as it recognizes the shape of an object, if we train a backpack, it will detect other backpacks of similar shape, even with different colors, i.e., the solution would generalize.

Currently, each ODDS unit costs around $314 (when Jetson Tk1 is used) to $624 (when Jetson TX1 is used) as it costs $80 for a Kinect v2, $44 for a Kinect adapter, $190 for a Jetson TK1, and $500 for a Jetson TX1. However, with a popularity of depth sensors and embedded GPUs, the price is declining rapidly, e.g., TI OPT8241 depth sensor costs $32 and Intel Movidius Neural Compute Stick costs $80, which requires a Raspberry Pi 3 that costs $35. The cost of installation is not more than installing a RGB camera that requires running a power cable.

ODDS can not detect an object if it is concealed in another opaque object. RGB camera based solutions also suffer from similar shortcomings. However, some of our proposed techniques could be useful in detecting objects if data is collected using a millimeter wave scanner or a backscatter X-ray machine that is used in security checking in airports. The accuracy of ODDS is good, but not great. However, object detection itself is very challenging and our performance is close to RGB camera based state of the art solutions with significantly less computing requirement and cost. We realize that our model may not work detecting actual guns as we used nerf guns for training and model construction. However, training with real guns with the same algorithm will enable us detecting real guns.

7 CONCLUSIONS

In this paper, we present a novel system called ODDS, which is thefi rst object detector that (i) uses only depth data, (ii) runs on an embedded GPU, and (iii) performs the entire processing in real-time. Our solution is much less privacy invasive than using an RGB camera and significantly cheaper than using a powerful GPU while maintaining comparable accuracy with the state of the art solutions. In the future, object detection and tracking could be a useful primitive in many smart building applications to enhance safety, security, productivity of the occupants and to improve the energy efficiency of the building. ODDS takes an important step towards realizing that vision.

8 ACKNOWLEDGEMENT

We thank Amit K. Roy-Chowdhury for many helpful comments. We would also like to thank the anonymous reviewers for the insightful suggestions. This work was supported, in part, by DOE grant DE-EE0007682. The opinions expressed here are those of the authors and do not necessarily reflect the views of the DOE.

REFERENCES

[1] 2017. Microsoft Kinect for XBOX One, https://www.xbox.com/en-US/xbox-one/accessories/kinect. (2017).

[2]2017. NVidia Jetson TX1 dev kit, https://developer.nvidia.com/embedded/buy/jetson-tx1-devkit. (2017).

[3] 2018. Building height. https://en.wikipedia.org/wiki/Storey. (2018).

[4] 2018. ODDS dataset. https://doi.org/10.5281/zenodo.1163770. (2018).

[5] Cesare Alippi, Maurizio Bocca, Giacomo Boracchi, Neal Patwari, and Manuel Roveri. 2016. RTI Goes Wild: Radio Tomographic Imaging for Outdoor People Detection and Localization. (2016).

[6] Jose M Alvarez and Mathieu Salzmann. 2016. Learning the number of neurons in deep networks. In *NIPS*. 2270–2278.

[7] A. Ashok, C. Xu, T. Vu, M. Gruteser, R. Howard, Y. Zhang, N. Mandayam, W. Yuan, and K. Dana. 2015. Low-Power Radio-Optical Beacons for In-View Recognition. In *VTC*.

[8] Boris Babenko, Ming-Hsuan Yang, and Serge Belongie. 2009. Visual tracking with online multiple instance learning. In *CVPR*. 983–990.

[9] Dan Banica and Cristian Sminchisescu. 2015. Second-order constrained parametric proposals and sequential search-based structured prediction for semantic segmentation in RGB-D images. In *CVPR*. 3517–3526.

[10] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. 2006. Surf: Speeded up robust features. In *ECCV*. Springer, 404–417.

[11] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *ICML*. ACM, 41–48.

[12] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. 2011. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning* 3, 1 (2011), 1–122.

[13] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. 2016. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs. *arXiv preprint arXiv:1606.00915* (2016).

[14] Yang Cong, Junsong Yuan, and Jiebo Luo. 2012. Towards scalable summarization of consumer videos via sparse dictionary selection. *IEEE Trans. on Multimedia* 14, 1 (2012), 66–75.

[15] Navneet Dalal and Bill Triggs. 2005. Histograms of oriented gradients for human detection. In *CVPR*, Vol. 1. IEEE, 886–893.

[16] Ehsan Elhamifar, Guillermo Sapiro, and Rene Vidal. 2012. See all by looking at a few: Sparse modeling forfi nding representative objects. In *CVPR*. IEEE, 1600–1607.

[17] Mark Everingham, Luc Van Gool, Chris Williams, John Winn, and Andrew Zisserman. 2009. The PASCAL Visual Object Classes (VOC) Dataset and Challenge. (2009).

[18] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks.. In *Aistats*, Vol. 9. 249–256.

[19] Saurabh Gupta, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. 2015. Indoor scene understanding with RGB-D images: Bottom-up segmentation, object detection and semantic segmentation. *IJCV* 112, 2 (2015), 133–149.

[20] Saurabh Gupta, Judy Hoffman, and Jitendra Malik. 2016. Cross modal distillation for supervision transfer. In *CVPR*. 2827–2836.

[21] Yihui He, Xiangyu Zhang, and Jian Sun. 2017. Channel Pruning for Accelerating Very Deep Neural Networks. *arXiv preprint arXiv:1707.06168* (2017).

[22] Andrew G Howard. 2013. Some improvements on deep convolutional neural network based image classification. *arXiv preprint arXiv:1312.5402* (2013).

[23] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. *arXiv preprint arXiv:1602.07360* (2016).

[24] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional architecture for fast feature embedding. In *ACM MM*. ACM, 675–678.

[25] Lu Jiang, Deyu Meng, Qian Zhao, Shiguang Shan, and A. G Hauptmann. 2015. Self-Paced Curriculum Learning.. In *AAAI*, Vol. 2. 6.

[26] Avinash Kalyanaraman, Dezhi Hong, Elahe Soltanaghaei, and Kamin Whitehouse. 2017. Forma Track: Tracking People Based on Body Shape. *IMWUT* 1, 3, Article 61 (Sept. 2017), 21 pages.

[27] Chitra R. Karanam and Yasamin Mostofi. 2017. 3D Through-wall Imaging with Unmanned Aerial Vehicles Using Wifi. In *IPSN*.

[28] Nacer Khalil, Driss Benhaddou, Omprakash Gnawali, and Jaspal Subhlok. 2017. SonicDoor: Scaling Person Identification with Ultrasonic Sensors by Novel Modeling of Shape, Behavior and Walking Patterns. In *BuildSys*.

[29] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS*. 1097–1105.

[30] Vadim Lebedev and Victor Lempitsky. 2016. Fast convnets using group-wise brain damage. In *CVPR*. 2554–2564.

[31] Yann LeCun and Yoshua Bengio. 1995. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks* 3361, 10 (1995), 1–14.

[32] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. 2016. SSD: Single shot multibox detector. In *ECCV*. Springer, 21–37.

[33] David G Lowe. 1999. Object recognition from local scale-invariant features. In *ICCV*, Vol. 2. IEEE, 1150–1157.

[34] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. 2017. ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression. *arXiv preprint arXiv:1707.06342* (2017).

[35] Niluthpol Chowdhury Mithun, Rameswar Panda, and Amit K Roy-Chowdhury. 2016. Generating diverse image datasets with limited labeling. In *ACM Multimedia Conference*. ACM, 566–570.

[36] Hessam Mohammadmoradi, Sirajum Munir, Omprakash Gnawali, and Charles Shelton. 2017. Measuring People-Flow Through Doorways using Easy-to-Install IR Array Sensors. In *DCOSS*.

[37] Sirajum Munir, Singh Arora Arora, Craig Hesling, Juncheng Li, Jonathan Francis, Charles Shelton, Christopher Martin, Anthony Rowe, and Mario Berges. 2017. Real-Time Fine Grained Occupancy Estimation using Depth Sensors on ARM Embedded Platforms. In *RTAS*.

[38] Sirajum Munir, Le Tran, Jonathan Francis, Charles Shelton, Ripudaman Singh Arora, Craig Hesling, Matias Quntana, Anand Krishnan Prakash, Anthony Rowe, and Mario Berges. 2017. Demo: FORK: Fine grained Occupancy estimatoR using Kinect on ARM Embedded Platforms. In *BuildSys*.

[39] Shijia Pan, Ningning Wang, Yuqiu Qian, Irem Velibeyoglu, Hae Young Noh, and Pei Zhang. 2015. Indoor Person Identification Through Footstep Induced Structural Vibration. In *HotMobile*.

[40] S. Papaioannou, A. Markham, and N. Trigoni. 2016. Tracking People in Highly Dynamic Industrial Environments. (2016).

[41] Anastasia Pentina, Viktoriia Sharmanska, and Christoph H Lampert. 2015. Curriculum learning of multiple tasks. In *CVPR*. 5492–5500.

[42] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You only look once: Unified, real-time object detection. In *CVPR*. 779–788.

[43] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NIPS*. 91–99.

[44] Kalyan Roy and Joydeep Mukherjee. 2013. Image similarity measure using color histogram, color coherence vector, and sobel method. *IJSR* 2, 1 (2013), 538–543.

[45] Ivan Sikirić, Karla Brkić, and SinišaŠegvić . 2013. Classifying traffic scenes using the GIST image descriptor. *arXiv preprint arXiv:1310.0316* (2013).

[46] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[47] Richard Socher, Brody Huval, Bharath Bath, Christopher D Manning, and Andrew Y Ng. 2012. Convolutional-recursive deep learning for 3d object classification. In *NIPS*. 656–664.

[48] Shuran Song, Samuel P Lichtenberg, and Jianxiong Xiao. 2015. Sun rgb-d: A rgb-d scene understanding benchmark suite. In *CVPR*. 567–576.

[49] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going Deeper with Convolutions. In *CVPR*.

[50] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *CVPR*. 1–9.

[51] Iwan Ulrich and Illah Nourbakhsh. 2000. Appearance-based place recognition for topological localization. In *ICRA*, Vol. 2. IEEE, 1023–1029.

[52] Xiaolong Wang and Abhinav Gupta. 2015. Unsupervised learning of visual representations using videos. In *ICCV*. 2794–2802.

[53] Chenren Xu, Bernhard Firner, Robert S. Moore, Yanyong Zhang, Wade Trappe, Richard Howard, Feixiong Zhang, and Ning An. 2013. SCPL: Indoor Device-free Multi-subject Counting and Localization Using Radio Signal Strength. In *IPSN*.

[54] Shuochao Yao, Yiran Zhao, Aston Zhang, Lu Su, and Tarek Abdelzaher. 2017. DeepIoT: Compressing Deep Neural Network Structures for Sensing Systems with a Compressor-Critic Framework. In *SenSys*.

[55] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks?. In *NIPS*. 3320–3328.

[56] Matthew D Zeiler and Rob Fergus. 2014. Visualizing and understanding convolutional networks. In *ECCV*. Springer, 818–833.

[57] Pengyu Zhang, Bodhi Priyantha, Jie Liu, and Matthai Philipose. 2015. *Redesigning the Wearable Camera Pipeline to Detect Key Body Parts at Low Power*. Technical Report.

[58] Hao Zhou, Jose M Alvarez, and Fatih Porikli. 2016. Less is more: Towards compact cnns. In *ECCV*. Springer, 662–677.

[59] Hongyuan Zhu, Jean-Baptiste Weibel, and Shijian Lu. 2016. Discriminative multimodal feature fusion for rgbd indoor scene recognition. In *CVPR*. 2969–2976.