CountNet3D: A 3D Computer Vision Approach to Infer Counts of Occluded Objects

Porter Jenkins[†], Kyle Armstrong[‡], Stephen Nelson [†],
Siddesh Gotad [‡], J. Stockton Jenkins[†], Wade Wilkey[‡], Tanner Watts[§]
[†]Brigham Young University, [‡]Delicious AI, [§]University of Utah
[†]pjenkins@cs.by.edu, {sn258, sjenkin2}@byu.edu
[‡]{kyle.armstrong, sidd.gotad, wade.wilkey}@deliciousai.com, [§]u1090501@utah.edu

Abstract

3D scene understanding is an important problem that has experienced great progress in recent years, in large part due to the development of state-of-the-art methods for 3D object detection. However, the performance of 3D object detectors can suffer in scenarios where extreme occlusion of objects is present, or the number of object classes is large. In this paper, we study the problem of inferring 3D counts from densely packed scenes with heterogeneous objects. This problem has applications to important tasks such as inventory management or automatic crop yield estimation. We propose a novel regression-based method, CountNet3D, that uses mature 2D object detectors for finegrained classification and localization, and a PointNet backbone for geometric embedding. The network processes fused data from images and point clouds for end-to-end learning of counts. We perform experiments on a novel synthetic dataset for inventory management in retail, which we construct and make publicly available to the community. Our results show that regression-based 3D counting methods systematically outperform detection-based methods, and reveal that directly learning from raw point clouds greatly assists count estimation under extreme occlusion. Finally, we study the effectiveness of CountNet3D on a large dataset of real-world scenes where extreme occlusion is present and achieve an error rate of 11.01% .

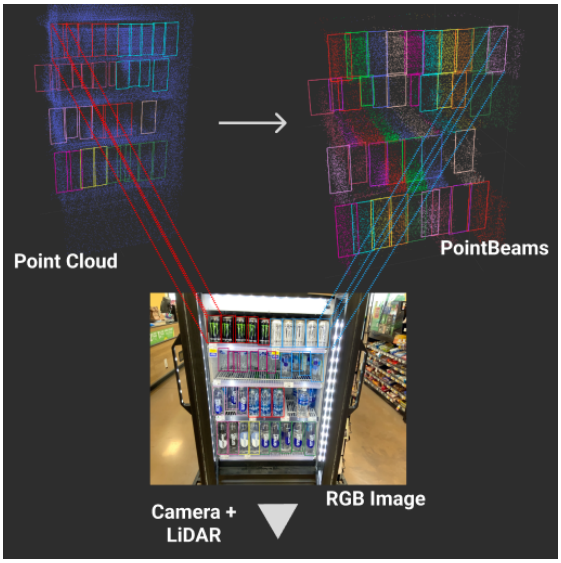

Figure 1: **Example of the 3D count inference problem**. (Best viewed in color) With an RGB camera and LiDAR sensor, we capture an image and a corresponding point cloud of a scene with occluded objects. The goal is to infer the total counts of each finegrained class. The problem is challenging because objects are densely spaced, and extremely occluded. Additionally, the objects may have the same geometric shapes, but have different class labels. Above we show a scene with an image, a point cloud with 2D detections lifted to 3D, and our PointBeam proposals

1. Introduction

Automatically identifying and counting densely spaced objects in 3D space is an important problem with many real-world applications. A system that is able to accurately identify and count objects can be used to streamline physical processes. For example, in physical retail and inventory management it can be challenging to know how many products are on a shelf at any given time. Any auditing of product counts must be done manually, which can be tedious and time consuming especially when the number of object classes is large. Other applications might include estimating agricultural crop yields [41][21] where a surveyor has to cover large farm lands and infer counts with sampling and interpolation. In either case, existing techniques cannot be easily used to automate the task.

In the current work, we seek to automatically infer counts of 3D objects that are densely spaced, and suffer from extreme occlusion. See Figure 1 for an example in a retail setting. In many cases, the objects are placed one in front of the other, making inference from visual input alone intractable. We propose to fuse visual information from images with 3D information from point clouds for end-to-end learning of 3D counts.

Recently, the computer vision community has devoted significant attention to inferring object counts in images. For instance, crowd counting is a well-studied problem with many robust methods for operating on single [14][42][17][44][19] or sequences [40] of images. Other work has tried to unify object counting methods by quickly adapting from crowds, to new domains such as wildlife, vehicles [11], and cancerous cells [20]. Existing studies primarily infer counts from 2D images and are not able to process 3D scene representations such as point clouds.

The 3D count inference problem has numerous technical challenges. First, objects are positioned in close proximity, and are to expected to significantly occlude more distant objects. We refer to this as the extreme occlusion problem. Such occlusion renders both detection and count inference from RGB images alone intractable because salient features of each occluded object are not clearly visible. Even simply applying powerful 3D object detectors is likely to yield poor performance due to severe occlusion of objects. Second, objects in a scene are heterogeneous and therefore need to be both classified and counted. Previous studies assume that all objects in an image or scene belong to the same class (e.g., all target objects are people). Realistic problems often involve classification and counting over a large set of object categories. Third, the lack of benchmark datasets and high cost of labeling 3D data has been too prohibitive to study the problem in the past. To the best of our knowledge no existing benchmark dataset exists for the 3D count inference problem.

In the current paper we seek to solve these problems with a novel, regression-based deep learning architecture that we call CountNet3D. Our proposed method processes multi-sensor data from RGB images and LiDAR point clouds and outputs finegrained object count estimates. We use a mature 2D object detector to identify and localize heterogenous objects from images, along with data fusion that occurs by lifting the detections into 3D space using the camera pose, camera intrinsic properties, and ray casting from SLAM output. We then segment the point cloud into smaller subspaces around the localized objects, constructing what we call PointBeams. PointBeam proposals use the shape of the 2D bounding box to facilitate very finegrained count estimation by reducing the search space to local neighborhoods around known objects. We use a PointNet backbone to learn geometric features for each PointBeam, followed by fully connected layers that predict the total number of objects within the PointBeam.

In our experiments we compare regression-based methods to state-of-the-art detection-based counting methods and demonstrate learning counts end-to-end greatly improves performance. CountNet3d achieves a 3.9% percentage error on our test set, which represents a 33.96% reduction in error compared to the most effective 3D object detector. Additionally, we compare the proposed PointBeams segmentation to global processing of point clouds and find that the PointBeams proposal generally improves the performance under extreme occlusion. We perform our experiments on a new synthetic dataset, called 3DBev24k, that contains LiDAR simulations of occluded objects on retail shelves, which make publicly available. Finally, we evaluate CountNet3d on a novel, real-world dataset comprised of 7.8k LiDAR scans of retail shelves and observe an error 11.01%, which outperforms all alternative 3D counting methods.

Our key contributions are summarized as follows:

- We propose a novel, regression-based framework for counting densely spaced objects in 3D, called CountNet3D
- We show that regression-based 3D counting methods outperform state-of-the-art 3D object detectors
- We apply CountNet3D to a real-world inventory management problem, and perform experiments on a large-scale dataset of simulated LiDAR scenes, which we make publicly available. We also manually collect 7.8k scans of real-world retail shelves and demonstrate CountNet3d outperforms alternative methods.

2. Related Work

Visual Object Counting Visual object counting with RGB and RGB-D images has been studied in many domains. Historically, crowd counting has received significant attention [14] from the computer vision community because issues like occlusion, poor illumination and perspective make it challenging. Broadly speaking, two primary strategies exist for counting objects from images: regression- and detection-based techniques. Mardsen et al. construct a convolutional neural network (CNN) architecture that operates on image patches to predict local object counts, which are then refined with a global CNN layer to predict total counts [20]. Other techniques leverage CNN's trained to predict density maps, which produce count estimates through density map integration [42][17][14]. Zhang et al. solve the multi-view crowd counting problem by estimating 3D scene-level density maps [40]. Other solutions exist that rely on videos [44] and graphs [19]. Finally, recent work tackles precise object detection in densely packed

retail scenes [7] which could be used to count object in 2D. Our work differs in that we propose to directly count objects depth-wise using lidar point clouds.

3D Object Counting Compared to image-based counting, research on 3D object counting is relatively thin. Recent work proposes a lidar-based, remote sensor system for counting pedestrian on sidewalks [15]. The system uses handcrafted point-cloud features and a watershed clustering algorithm to determine pedestrian counts. Other work studies automated crop counting and proposes a novel pipeline that relies on 3D constructions of stereo images, singular value decomposition (SVD), and sphere fitting for fast and accurate grape count inference [21]. A key difference between these studies and the current work is that we seek to infer counts directly from point clouds with end-to-end learning.

Deep Learning with Point Clouds Deep Learning architectures that operate on point clouds have been useful for a variety of tasks including autonomous driving, 3-D object detection, and classification. The key idea in many of these works is to learn features on raw points clouds or voxels. PointNet [26] is a seminal deep learning architecture that operates on a point cloud directly, without having to transform data into 3D voxel grids [25]. PointNet++ [25] builds upon PointNet by introducing hierarchical feature learning through local neighborhoods. Recent work [24, 37, 22] demonstrates that PointNets, along with a 3D viewing Frustum, can be trained to detect objects in 3D. Critically, Frustum PointNets assume that a single object lies in the viewing frustum and therefore cannot be applied to our problem. PointCNN [16] introduces a convolution operator to leverage spatially-local correlation in data represented densely in grids. Other methods, such VoxelNet [43] and PointPillars [13] voxelize the point cloud and perform 3D or 2D convolutions on the voxels. In general, many other works seek to construct mechanisms to use convolution-like operators on raw points, or voxels [33][34][8][31]. The primary difference between our method and these is our use of a regression-based deep learning architecture for learning object counts in 3D.

3. Problem Formulation

In the following section we formally define the 3D count inference problem. Suppose we have a set of n scenes $\mathcal{S} = \{s_i : 0 < i <= n\}$, where each scene, $s_i = \langle \mathcal{X}^{(i)}, z^{(i)} \rangle$, is defined as a tuple containing a set of RGB images, $\mathcal{X}^{(i)} = \{x_1, x_2, ...\}$, and a point cloud, z_i. A LiDAR sensor is assumed to be co-calibrated with an RGB camera. Additionally, we have a set of m object classes, $\mathcal{C} = \{c_j : 0 < j <= m\}$, that are observed across the scenes in $\mathcal{S}$. From the object set we can construct an observed set of objects, and their counts in each scene. Let $\mathbf{y}^{(i)}$ be a vector of observed object counts in scene, s_i. The label vector, $\mathbf{y}^{(i)}$, is a vector of non-negative values, $\mathbf{y}^{(i)} \in \{0, 1, 2, ...\}^m$, where each component, $y_j^{(i)}$, denotes the count of class, c_j, in scene, s_i.

Our primary goal is to learn a function, $f(\cdot)$ that inputs a set of images, $\mathcal{X}^{(i)}$, and a point cloud, z_i, and outputs the estimated count, $\hat{y}_j^{(i)}$, of each class, c_j, in scene s_i. In vector notation, the function we seek to learn is

$$\hat{\boldsymbol{y}}^{(i)} = f(\mathcal{X}^{(i)}, z^{(i)}; \boldsymbol{\theta}) \tag{1}$$

where $\boldsymbol{\theta}$ is a vector of parameters (e.g., weights of a neural network). Our key hypothesis is that an end-to-end function, $f(\cdot)$, can be learned to map from a set of raw images and point clouds to 3D object counts.

4. CountNet3D

Due to the challenges of extreme occlusion, applying 3D object detectors to the count inference problem is very challenging and leads to poor accuracy. To solve this, we propose CountNet3D, a multi-modal deep architecture for inferring counts of densely spaced 3D objects.

As shown in Figure 2, CountNet3D is composed of two primary modules: PointBeam proposals to fuse data from images and LiDAR point clouds, and a count estimation module to process to the PointBeams and output 3D counts. In the following sections we introduce each component in detail, and provide implementation details relevant to our experiments.

4.1. PointBeams

Images contain rich information about objects in a 3D scene. Due to their high resolution format, they are particularly useful for localization and classification. This is especially true when making high-dimensional, fine grained classifications (e.g., species of a bird [36] [4], or distinguishing between different products [6] of similar shapes). To extract 2D detections from the image set, $\mathcal{X}^{(i)} = \{x_1, x_2, ...\}$, we use a YOLOv5 [12] model. We choose the YOLO detector because it allows for accurate predictions, and real-time processing of a sequence of RGB images. However, it's important to point out that CountNet3D is agnostic to the choice of 2D object detector. We pretrain our YOLO object detector on the COCO benchmark [18] and finetune on a custom inventory dataset.

4.1.1 Proposing PointBeams

One of the key challenges of the 3D count inference problem is that objects can be very crowded, creating severe occlusion. Given the low resolution nature of 3D sensors such as LiDAR, it has been shown to be very beneficial to reduce the search space in 3D object detection by proposing a 3D

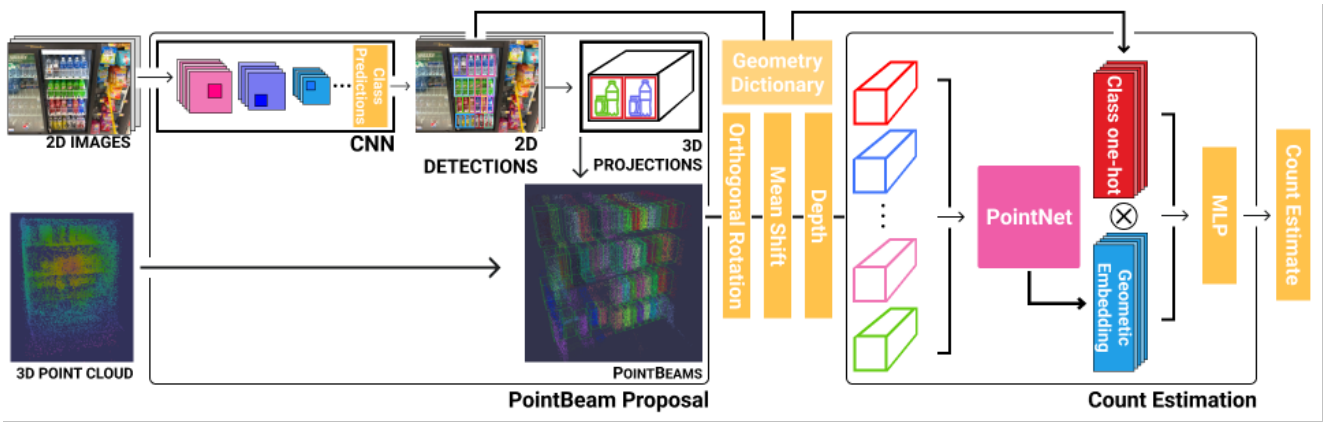

Figure 2: **Overview of CountNet3D**. (Best viewed in color). Our model inputs a set of images and a point cloud. The images are passed through a 2D CNN-based object detector, and the resulting bounding boxes are lifted into 3D to form the PointBeam proposals. Each beam is translated into a canonical subspace by applying an orthogonal rotation, mean shifting, and computing depth features, which results in an $(M \times 8)$ tensor. The PointBeams are fed into a PointNet backbone, which outputs a geometric embedding. Simultaneously, we use a geometric dictionary to look up a geometric class, and get a one-hot tensor, from the finegrained 2D detections. Finally, the one-hot tensor and geometric embeddings are concatenated and fed into a set of fully connected layers, which output a real-valued count estimated for each PointBeam.

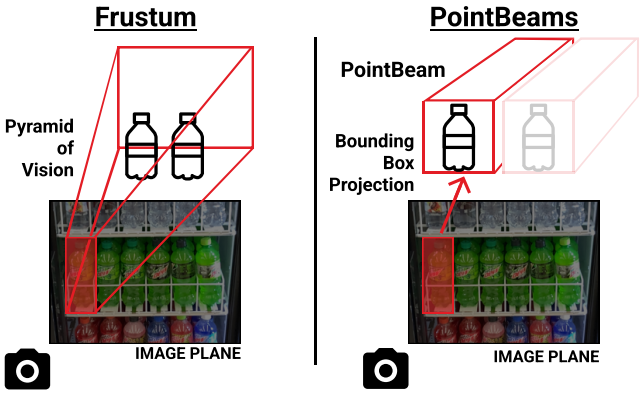

Figure 3: **Difference between a viewing Frustum and PointBeams.** Frustum PointNets [24][22] create a pyramid of vision, where the field of view enlarges as the distance from the camera increases. PointBeams project the 2D bounding box onto the location of the detected object and extend it using the normal vector with respect to the image plane. The field of view is constant as the distance increases. PointBeams offer a more focused and localized view of a set of densely spaced objects. Additionally, Frustum PointNets [24] cannot be applied when multiple target objects lie in a frustum.

viewing frustum [24][22]. The viewing frustum acts to reduce the dimensionality of the point cloud, and focus the 3D backbone on subspaces of known objects only.

Building upon this idea, we propose a novel method to subdivide the point cloud and focus our 3D regressor, which we call PointBeams. PointBeams project a 2D bounding box onto the estimated location $\langle x, y, z \rangle$ of a detected object, and compute a rectangular prism by extending the 2D plane (w, h) along the normal vector up to a pre-specified depth, d. The nomal vector is computed with respect to the image plane. In contrast to a viewing frustum, PointBeams have the property that the field of view remains constant as the distance from the camera increases. This is very useful in densely packed scenes where multiple frusta will begin to overlap. See Figure 3 for a visual explanation of the differences between viewing frusta and PointBeams. We are able to lift the 2D bounding boxes into 3D using the camera pose, camera intrinsic properties, and ray casting [5].

4.1.2 Representation of PointBeams

Once a PointBeam has been proposed for each object detected in the image layer, we map each point in the point cloud to the correponding PointBeam. Points that do not lie in a beam are discarded. We find that proper representation of points within each beam is critical (Table 3).

Orthogonal rotation: To improve the rotation invariance of our network, we rotate each beam around the vertical axis such that the beam is orthogonal to the center axis. Consequently, we can propose beams for objects of arbitrary poses and orientations.

Mean Shift: To improve the translation invariance of our network, we mean shift each PointBeam proposal by subtracting each point by the mean of all the points in the proposal. The resulting local coordinate representation pro-

Figure 4: **Example predictions from CountNet3D on real-world data**. (Best viewed in color with zoom in). The PointBeam proposals, the ground truth (GT), and the predicted counts (pred) are shown. We also display the scene-level MAPE. Each beam is given a randomly generated color to highlight the beam regions. We only display the global $\langle x, y, z \rangle$ for visualization clarity. We observe that under extreme occlusion, where both 2D and 3D object detectors are likely to fail, CountNet3D is able to accurately predict the object counts. See supplementary materials for additional image/video examples.

duces proposals that are consistently centered and scaled. Consequently, if two objects fall in different positions in the scene, the network doesn't have to learn new geometric features from scratch each time.

Beam Depth: Intuitively, the number of objects in a proposal is correlated with how close to the front or back each object is. To directly model this intuition, we compute depth features for each point in the beam. We measure the distance in the forward dimension to the most distant point within the beam, and the distance to the nearest point in the beam. Assuming a y forward coordinate space, for each point $\langle x_i, y_i, z_i \rangle$, we calculate $D = \langle y_{max} - y_i, y_{min} - y_i \rangle$

We concatenate the global $\langle x_g, y_g, z_g \rangle$ coordinates, the local $\langle x_l, y_l, z_l \rangle$ coordinates (obtained via mean shifting), and the depth features into an 8 channel input point cloud. We report on the effects of these point representations in Table 3.

4.2. Count Estimation

Once the PointBeams are computed and transformed, they are then stacked into a single tensor and fed into a PointNet model . Simultaneously, we construct a one-hot vector denoting geometry type, which is concatenated with the geometric embedding tensor. We extract the predicted class, $\hat{c}$ label from the image detection and use a pre-defined geometry dictionary that maps the finegrained class, $\hat{c}$, (e.g., coca-cola_20oz_bottle) to a coarse geometric type, g (e.g., 20ozBottle). Since the PointBeams condition on the predicted class from the image, the subsequent layers only need to reason about geometric shapes present in the beam. This is a necessary dimensionality reduction step that reduces the set of candidate classes from 3000+ to five distinct geometry types. If the number of object classes is relatively small, this step can be skipped entirely and the predicted class can be used directly in the one-hot tensor. More details about this dictionary are provided in the appendix.

After concatenating the geometric embedding and the one-hot tensor, the resulting tensor is fed into a set of 5 fully connected layers with dimensions $[512, 256, 64, 64, 64]$. ReLU activations and batch normalization are used after each layer. The output of the count estimation network is a single scalar $\hat{y}$. Since the PointBeams are stacked and passed through the count estimation network, the output tensor has dimensions $(B \times N \times 1)$, where B is the batch size and N is the total number of beams in each batch. From the output tensor we can easily get a tuple $\langle \hat{c}_{ij}, \hat{y}_{ij} \rangle$, which describes the predicted finegrained class and the total count within each PointBeam. Finally, we sum the predicted counts within each class to get class-level count estimates.

One simplifying assumption implicit in the design of CountNet3D is what we refer to as the *locality assumption*: objects that are densely spaced together tend to be of the same class. Specifically, occluded objects that fall within a beam cast by the bounding box plane are assumed to be of the same class as those that are easily visible from an image. This is a reasonable assumption in many applications including inventory management and crop yield estimation.

4.3. Loss Function

We train CountNet3D for end-to-end learning of counts using a simple squared error loss function:

$$\mathcal{L}_{scene} = \Sigma_j (y_j - \hat{y}_j)^2 \tag{2}$$

We assume that each scene, s_i, contains a set of k_i object detections. Consequently, the loss function we optimize is the average squared error over a batch of scenes:

$$\mathcal{L} = \frac{\Sigma_i \Sigma_j (y_{i,j} - \hat{y}_{i,j})^2}{N} \tag{3}$$

	3DBev24k	Real-world
Train	18,984	6,259
Test	4,820	812
Val	-	811
Finegrained Classes	4,074	359
Geo. Classes	5	21
μ	3.12	8.15
σ	1.59	7.84
Max	24	75

Table 1: Dataset Summary

where i indexes scene, s_i in a batch, and j indexes each PointBeam, b_j. The denominator, $N = \Sigma_i k_i$, is simply the total number of PointBeams proposed across all the scenes in the batch.

4.4. Implementation Details

Point clouds have a variable size, due to the variable number of points captured by the LiDAR. Additionally, each scene, s_i, contains a variable number of k_i object detections. Both of these issues make mini-batch training with stacked tensors difficult. To solve this problem, we impose a maximum points parameter, M, on each PointBeam. If the number of points that fall in the beam is greater than M, then we downsample the points to be of size M. Otherwise, we zero-pad the PointBeam tensor. In our experiments we set $M = 1024$. This operation produces tensors of a fixed size that can be easily stacked into a mini-batch [27]. The resulting point cloud tensor has dimensions $B \times N \times M \times C$, where B is the batch size, and C is the number of channels (Table 3). We use an image resolution of 640×480 in the 2D detection layer.

We preprocess the entire scene by mean shifting and transforming into the unit ball by dividing each point by the maximum norm of all the points. This puts all the point clouds at the origin with a normalized point clouds range in [-1, 1] in $\langle x, y, z \rangle$. The normalized point clouds are then fed into the PointBeam proposal layer. When proposing PointBeams, we introduce two additional hyperparameters to control the shape and size of the beams. First, ϵ, perturbs the dimensions of the 2D bounding box plane: $\langle w*(1+\epsilon), h*(1+\epsilon) \rangle$. This allows CountNet3D to capture points around the boundary of the target object. Second, δ, determines the depth of each beam. In our experiments, we set $\epsilon = 0.05$ and $\delta = 0.6$

5. Experiments

In this section we define our experimental setup. We perform experiments on two datasets, one synthetic and one real-world. Both of the datasets contain scenes of beverages on shelves.

5.1. Datasets

3DBev24k: In this section, we describe our new dataset, 3DBev24k. The data are comprised of scenes built using the popular 3D graphics software Blender [3]. We manually construct scenes of retail shelves with object placement similar to real-world scenes. We add variance to the data during simulation by 1) randomizing LiDAR physics parameters 2) masking objects out of the scene. For each scene, the simulation process then outputs a point cloud using Blensor [9], and a variety of annotations including class counts, bounding boxes, and semantic segmentation labels. The classes of the objects are organized hierarchically and correspond to products typically seen in beverage retail. For each object we provide a finegrained class (ie., 'coca_cola_20oz_bottle') and a geometric class (i.e., '20ozBottle'). A canonical train/test split is also defined with 18,984 train examples and 4,820 test examples. See the supplementary materials for a detailed description of the simulator and example scenes. Summary statistics for 3DBev24k are provided in Table 1. This dataset is open source and can be accessed via online supplement.

Real-world Data: We also perform an experiment with a proprietary real-world dataset to evaluate the effectiveness of CountNet3d on complicated, physical scenes. This dataset is comprised of 7,882 annotated examples. Summary statistics are reported in Table 1. Using a custom iOS mobile application on iPhone 12/13 and iPad Pro 4th generation with built-in LiDAR [2], we capture images and point clouds of real-world scenes with strong occlusion in a retail setting. These scenes are also annotated with finegrained class counts by experts. Due to the difficulty of labelling point clouds of very crowded scenes with 3D bounding boxes, we are unable to compare detection-based methods on the real-world data. We pretrain all models on the synthetic data, and finetune on the smaller, real-world dataset.

5.2. Evaluation

We evaluate all methods using Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), and Mean-squared Error (MSE). For all three metrics, lower values correspond to lower error and more accurate count estimation.

5.3. Settings

Our experiments seek to answer three primary questions. First, what is the performance of regression-based methods compared to detection-based methods for solving the 3D count inference problem? Second, what is the effect of point representation on CountNet3D performance? Third, how well does CountNet3D perform on real-world retail scenes with extreme occlusion? We answer these questions in the following three sections.

Table 2: Evaluation on a synthetic, test dataset. Lower is better

		Method	Input	MAE	MAPE	MSE
Detection	Global	YOLO [28]	RGB	2.5370	0.7235	9.4824
		SECOND [38]	3D	2.7245	0.9396	9.9718
	PointBeams	SECOND [38]	RGB+3D	2.175	0.6775	7.1816
		PointPillars [13]	RGB+3D	1.5234	0.7659	3.6412
		VoteNet [23]	RGB+3D	1.2768	0.4938	2.9704
		PIXOR [39]	RGB+BEV	1.1906	0.3786	3.3334
		YOLO [28]	RGB+BEV	0.3649	0.1274	0.7776
Regression	PointBeams	VGG-16 [32]	RGB+BEV	0.2959	0.0892	0.6397
		MobileNetV2 [29]	RGB+BEV	0.3228	0.1104	0.6963
		ResNet-18 [10]	RGB+BEV	0.2773	0.1049	0.3399
		CountNet3D (ours)	RGB+3D	**0.1134**	**0.0390**	**0.1268**

Table 3: Effect of point cloud representation on CountNet3D (synthetic data). Lower is better

Channels	Global	Local	Distance	MAE	MAPE	MSE
3	✓	✗	✗	0.5624	0.2195	0.9292
3	✗	✓	✗	0.3690	0.1232	0.6895
6	✓	✓	✗	0.2031	0.0722	0.2250
5	✗	✓	✓	0.2167	0.0720	0.2571
8	✓	✓	✓	**0.1134**	**0.0390**	**0.1268**

5.4. Regression- vs. Detection-based Methods

The major hypothesis of this work is that regression-based approaches are superior to detection-based approaches for counting densely spaced objects in 3D. This hypothesis is informed by two theoretical considerations. First, 3D detection methods are designed to classify and localize objects. Amodal localization of objects in 3D space is very challenging, especially as the degree of occlusion increases. In the case of the Kitti benchmark leaderboard [1], nearly all detection-based methods degrade as the level of occlusion increases. In the context of count inference, we believe we can bypass the statistically difficult step of estimating a vector of positional coordinates, and instead estimate a scalar count value. Second, using detection-based methods to solve the counting problem suffers from a misaligment between our objective (counting) and our loss function (detection). By aligning our objective (counting) with our loss (squared error of counts), we can take advantage of the benefits of end-to-end learning.

5.4.1 Object Detectors

We train one 2D, and multiple 3D object detectors to classify and localize objects in each scene. YOLOv5 [28][12] is trained to detect objects from images only. The 3D detectors are trained to detect each of the geometric types from point clouds. We compare CountNet3D to PIXOR [39], SECOND [38] [35], PointPillars [13] [22] [35] and VoteNet [23]. For each object detector, we estimate the locations of each object, and sum over the bounding boxes to get a per-class count.

5.4.2 Bird's Eye View (BEV) Regressors

In addition to CountNet3D we also implement and test a novel pipeline for using 2D convolutions as regressors. We project each PointBeam proposal to a 200×200 BEV image (Figure 5) with a resolution parameter of .01. This point cloud representation intuitively simplifies the counting problem into an easier perceptual problem (counting basic shapes such as rings), and facilitates the use of mature 2D CNN architectures. We train four baseline CNN's on the bird's eye view projections of each PointBeam proposal: VGG-16 [32] , ResNet-18 [10], and MobileNetV2 [29], and YOLOv5 [28][12].

5.4.3 Results

We report our results in Table 2 and have three primary observations. First, in all cases, regression-based techniques significantly outperform 3D detection-based methods. Even the weakest regressor, a MobileNetV2 applied to a BEV projection, outperforms PIXOR, the most effective 3D ob-

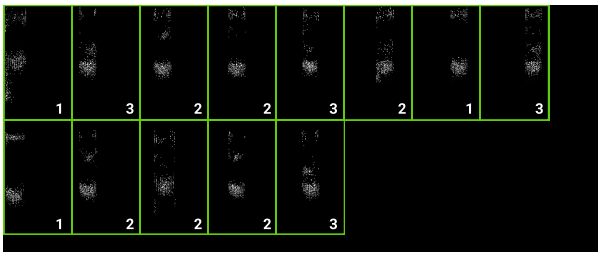

Figure 5: Example bird's eye view (BEV) representation applied to the PointBeam proposals, where each cell is a separate proposal, along with its ground truth count. This simplified representation outperforms 3D object detectors, but cannot match the performance of CountNet3D

ject detector. Interestingly the YOLOv5 model trained on the BEV images dramatically outperforms the 3D detectors, suggesting the effective representation of BEV images. CountNet3D yields the best performance of all, outperforming PIXOR by 33.96% and VGG-16 by 5.01% in MAPE. Second, Table 2 suggests that full 3D is generally better than a BEV image when doing regression. CountNet3D is the only regression method that leverages full 3D information from the point clouds, and reduces MAPE by 7.14% - 5.01% compared to the BEV regressors. In Section 5.5, we discuss why this is the case. Third, PointBeams is a necessary step to solve the 3D counting problem. We also train a SECOND detector on the global point cloud (no PointBeam proposals), and match the 3D geometric classes to the fine-grained classes from the images using a nearest neighbors method. We see that both YOLO and SECOND with PointBeams proposals are more accurate than the variants trained on the full point cloud. Recent work studying Frustum-based detection has shown similar results [22, 37, 24, 30]. Due to vertical occlusion, we cannot apply PointPillars and PIXOR to the global point cloud.

5.5. Effect of PointBeam Representation

Proper representation of points within each PointBeam is crucial. In Table 3 we explore the effect that point representation has on prediction error. We see that normalizing the point cloud within each beam by subtracting the centroid (*local*) has a large effect on error reduction. This creates a canonical subspace across all PointBeams and improves translation invariance. We find another large effect when the global and local coordinates are used together. Finally, we find another modest error reduction when using the depth features. The 8 channel point representation allows the model to make reasonable predictions even in difficult cases where objects are extremely occluded and very few points cover some objects. The CNN-based regressors have no mechanism to handle such cases.

Table 4: Evaluation of regression-based methods on a real-world, test dataset. Lower is better

Method	MAE	MAPE	MSE
VGG-16 [32]	0.9400	0.2462	1.5030
MobileNetV2 [29]	0.4473	0.1476	0.4808
ResNet-18 [10]	0.4474	0.1454	0.4749
CountNet3D (ours)	**0.3500**	**0.1101**	**0.4739**

5.6. Evaluation on Real-world Data

We report results in Table 4. Our primary observation is that CountNet3D outperforms the BEV-based regression methods across all three evaluation criteria. In particular, CountNet3D demonstrates significant performance increase in the average case (MAE, MAPE), and yields a 3.5% reduction in MAPE compared to ResNet. While still superior, the reduction in MSE is slightly lower compared to ResNet and MobileNet. We believe this is likely due to scenes where the point cloud only covers a single object in the front of the PointBeam, and the model can't discriminate between many objects (strong occlusion) and a single object (no occlusion). Because of the squared error term, MSE aggressively penalizes mistakes on these challenging examples.

5.7. Qualitative Analysis

An example from our real-world test dataset is visualized in Figure 4. The scene has a variable number of ground truth object counts (in the range 2-6). We observe that even if LiDAR points are unable to cover each individual object, and point density decreases for more occluded objects, our model is still able to generalize to a correct count prediction. Table 3 suggests this is likely due to the addition of the local coordinates and depth features. Detectors tend to fail on these examples.

6. Conclusion

In this paper, we proposed CountNet3D, which is a novel, regression-based deep learning approach to end-to-end learning of object counts. We construct and make publicly available a benchmark dataset called 3DBev24k to encourage further exploration of the 3D count inference problem. We show that CountNet3D significantly outperforms state-of-the-art 3D object detectors, and has good performance in real-world settings.

Acknowledgements

We are grateful to our colleagues for their valuable contributions: Zihan Zhou, Michael Selander, Andrew Merrill, Brad Curtis, Gary Ekker, Isaac Tai and Michael Holland.

References

[1] The kitti vision benchmark suite: Leaderboard.

[2] Apple introduces iphone 12 pro and iphone 12 pro max with 5g, 2020.

[3] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018.

[4] Abhimanyu Dubey, Otkrist Gupta, Pei Guo, Ramesh Raskar, Ryan Farrell, and Nikhil Naik. Training with confusion for fine-grained visual classification. *CoRR*, abs/1705.08016, 2017.

[5] Andrew S. Glassner. *Introduction to Ray Tracing*. Morgan Kaufmann, 1989.

[6] Eran Goldman, Roei Herzig, Aviv Eisenschtat, Jacob Goldberger, and Tal Hassner. Precise detection in densely packed scenes. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5222–5231, 2019.

[7] Eran Goldman, Roei Herzig, Aviv Eisenschtat, Oria Ratzon, Itsik Levi, J. Goldberger, and Tal Hassner. Precise detection in densely packed scenes. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5222–5231, 2019.

[8] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 3d semantic segmentation with submanifold sparse convolutional networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9224–9232, 2018.

[9] Michael Gschwandtner, Roland Kwitt, Andreas Uhl, and Wolfgang Pree. Blensor: Blender sensor simulation toolbox. In *Proceedings of the 7th International Conference on Advances in Visual Computing - Volume Part II*, ISVC'11, page 199–208, Berlin, Heidelberg, 2011. Springer-Verlag.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[11] Meng-Ru Hsieh, Y. Lin, and W. Hsu. Drone-based object counting by spatially regularized regional proposal network. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 4165–4173, 2017.

[12] Glenn Jocher, Alex Stoken, Ayush Chaurasia, Jirka Borovec, NanoCode012, TaoXie, Yonghye Kwon, Kalen Michael, Liu Changyu, Jiacong Fang, Abhiram V, Laughing, tkianai, yxNONG, Piotr Skalski, Adam Hogan, Jebastin Nadar, imyhxy, Lorenzo Mammana, AlexWang1900, Cristi Fati, Diego Montes, Jan Hajek, Laurentiu Diaconu, Mai Thanh Minh, Marc, albinxavi, fatih, oleg, and wanghaoyang0106. ultralytics/yolov5: v6.0 - YOLOv5n 'Nano' models, Roboflow integration, TensorFlow export, OpenCV DNN support, Oct. 2021.

[13] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12697–12705, 2019.

[14] V. Lempitsky and Andrew Zisserman. Learning to count objects in images. In *NIPS*, 2010.

[15] Asad Lesani, Ehsan Nateghinia, and Luis Miranda-Moreno. Development and evaluation of a real-time pedestrian counting system for high-volume conditions based on 2d lidar. *Transportation Research Part C-emerging Technologies*, 114:20–35, 2020.

[16] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. Pointcnn: Convolution on $\mathcal{X}$-transformed points, 2018.

[17] Dongze Lian, J. Li, Jia Zheng, Weixin Luo, and Shenghua Gao. Density map regression guided detection network for rgb-d crowd counting and localization. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1821–1830, 2019.

[18] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[19] Ao Luo, F. Yang, X. Li, Dong Nie, Zhicheng Jiao, Shangchen Zhou, and Hong Cheng. Hybrid graph neural networks for crowd counting. In *AAAI*, 2020.

[20] Mark Marsden, Kevin McGuinness, S. Little, Ciara E. Keogh, and N. O'Connor. People, penguins and petri dishes: Adapting object counting models to new visual domains and object types without forgetting. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8070–8079, 2018.

[21] Anjana K Nellithimaru and George A. Kantor. Rols : Robust object-level slam for grape counting. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 2648–2656, 2019.

[22] Anshul Paigwar, David Sierra-Gonzalez, Özgür Erkent, and Christian Laugier. Frustum-pointpillars: A multi-stage approach for 3d object detection using rgb camera and lidar. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, pages 2926–2933, October 2021.

[23] Charles R. Qi, Or Litany, Kaiming He, and Leonidas J. Guibas. Deep hough voting for 3d object detection in point clouds. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[24] Charles R. Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J. Guibas. Frustum pointnets for 3d object detection from rgb-d data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[25] C. R. Qi, H. Su, Kaichun Mo, and L. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 77–85, 2017.

[26] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017.

[27] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3d deep learning with pytorch3d. *arXiv:2007.08501*, 2020.

[28] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[29] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[30] Xiaoke Shen and Ioannis Stamos. Frustum voxnet for 3d object detection from rgb-d or depth images. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, March 2020.

[31] Shaoshuai Shi, Chaoxu Guo, Li Jiang, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. Pv-rcnn: Point-voxel feature set abstraction for 3d object detection. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10526–10535, 2020.

[32] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2015.

[33] Hang Su, Varun Jampani, Deqing Sun, Subhransu Maji, Evangelos Kalogerakis, Ming-Hsuan Yang, and Jan Kautz. Splatnet: Sparse lattice networks for point cloud processing, 2018.

[34] Maxim Tatarchenko, Jaesik Park, Vladlen Koltun, and Qian-Yi Zhou. Tangent convolutions for dense prediction in 3d. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3887–3896, 2018.

[35] OpenPCDet Development Team. Openpcdet: An open-source toolbox for 3d object detection from point clouds. `https://github.com/open-mmlab/OpenPCDet`, 2020.

[36] Grant Van Horn, Steve Branson, Ryan Farrell, Scott Haber, Jessie Barry, Panos Ipeirotis, Pietro Perona, and Serge Belongie. Building a bird recognition app and large scale dataset with citizen scientists: The fine print in fine-grained dataset collection. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 595–604, 2015.

[37] Zhixin Wang and Kui Jia. Frustum convnet: Sliding frustums to aggregate local point-wise features for amodal 3d object detection. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1742–1749. IEEE, 2019.

[38] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors*, 18:3337, 10 2018.

[39] Bin Yang, Wenjie Luo, and Raquel Urtasun. Pixor: Real-time 3d object detection from point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[40] Qi Zhang and Antoni B. Chan. 3d crowd counting via multi-view fusion with 3d gaussian kernels. *ArXiv*, abs/2003.08162, 2020.

[41] Qian Zhang, Yeqi Liu, Chuanyang Gong, Y. Chen, and HuiHui Yu. Applications of deep learning for dense scenes analysis in agriculture: A review. *Sensors (Basel, Switzerland)*, 20, 2020.

[42] Yingying Zhang, Desen Zhou, Siqin Chen, Shenghua Gao, and Yi Ma. Single-image crowd counting via multi-column convolutional neural network. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 589–597, 2016.

[43] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4490–4499, 2018.

[44] Z. Zou, Huiliang Shao, Xiaoye Qu, Wei Wei, and Pan Zhou. Enhanced 3d convolutional networks for crowd counting. *ArXiv*, abs/1908.04121, 2019.