Unlocking the potential of 3D Object Detection with bounding box estimation

Dhyey Manish Rajani, Rahul Kashyap Swayampakula, Surya Pratap Singh
{drajani, rahulswa, suryasin}@umich.edu

Abstract—**We introduce a technique for detecting 3D objects and estimating their position from a single image. Our method is built on top of a similar state-of-the-art technique [1], but with improved accuracy. The approach followed in this research first estimates common 3D properties of an object using a Deep Convolutional Neural Network (DCNN), contrary to other frameworks that only leverage centre-point predictions. We then combine these estimates with geometric constraints provided by a 2D bounding box to produce a complete 3D bounding box. The first output of our network estimates the 3D object orientation using a discrete-continuous loss [1]. The second output predicts the 3D object dimensions with minimal variance. Here we also present our extensions by augmenting light-weight feature extractors and a customized multibin architecture. By combining these estimates with the geometric constraints of the 2D bounding box, we can accurately (or comparatively) determine the 3D object pose better than our baseline [1] on the KITTI 3D detection benchmark [2].**

I. Introduction

The detection of 3D objects is crucial in robotics where decisions and interactions with objects in the real world are required. This involves identifying both the position and dimensions of an object from an image. While 2D detection algorithms have been successful in handling variations in viewpoint and clutter, accurately detecting 3D objects remains a difficult problem. Previous attempts to combine pose estimation with object detection have primarily focused on estimating viewpoint by dividing it into discrete categories that can be trained, based on the fact that an object's appearance changes with viewpoint [3]. In situations where full 3D pose estimation is not possible, alternative approaches involve sampling and scoring hypotheses using contextual and semantic cues [4].

This work introduces a method to estimate the position & dimensions of a 3D object's bounding box using a 2D bounding box and geometric constraints. Instead of direct pose regression, first the orientation & dimensions are estimated and then estimates are combined with geometric object constraints for 3D pose. Secondly, our approach even though is similar to [1], but we are able to achieve higher accuracy by replacing the original backbone network, i.e. VGG-19, with MobileNet and EfficientNet. After separate integration of these feature extractors, all the models were trained separately from scratch.

To enhance [5], DCNN is trained to estimate the orientation & dimensions of an object's 3D bounding box to constraint its the projection to fit within the 2D detection window, the method recovers the translation and the full 3D bounding box of the object. Despite its simplicity, the method builds upon several important insights. One of these is the novel approach introduced in [1], called MultiBin discrete-continuous formulation, which is used for orientation regression and is shown to outperform traditional L2 loss. Additionally, regressing to vehicle dimensions is effective in constraining the 3D box, since these dimensions have low variance and stable estimates.

Our method is evaluated on the KITTI 3D Object detection dataset [2], where we compare our estimated 3D boxes to the results of Mousavian et al [1]. Our method is evaluated on the KITTI 3D Object detection dataset [2], where we compare our estimated 3D boxes to the results of Mousavian et al [1]. Although the official benchmark for 3D bounding box estimation only evaluates the 3D box orientation estimate, we have used three additional performance metrics to assess accuracy: the distance to the box's center, the distance to the center of the nearest bounding box face, and the total amount of overlap between the bounding box and the ground truth box using the 3D Intersection over Union (3D IoU) score. We demonstrate that, with sufficient training data, our method outperforms the state-of-the-art algorithm on all of these 3D metrics. We demonstrate that, with sufficient training data, our method surpasses state-of-the-art algorithm on all 3D metrics.

In summary, our paper makes several key contributions. First, we develop the whole 3D pose estimation environment and core algorithm [1] from ground-up in Pytorch. This task is undertaken since the original paper has no official code implementation and other implementations on the web either have completely corrupted observations or have incorrect implementation of the 3D pose estimator. Second, we train the neural networks from scratch by internal transfer and check-pointing of weights due to computing limitations. Third, we show the rationality & effectiveness of using light-weight feature extractors instead of conventional ones by outperforming [1] on the KITTI object detection metrics. Finally, we combine these extractors with our customized multibin architectural extension to prove our framework's efficacy against [1].

II. Related Works

In the past, people have looked at the problem of figuring out the position and orientation of an object in 3D space using only a single 2D image. This is called the ”6 degrees of freedom pose estimation problem”. One way to solve this problem is by using a mathematical technique called the perspective n-point problem (PnP), which involves matching 2D points in the image with their corresponding 3D points in a model of the object. There are different ways to approach this problem, including using closed-form or iterative solutions [6], as well as constructing 3D models of the object and finding the best match between the model and the image [7] [8].

New datasets [2] [9] [10] [11] have made it possible to extend 3D pose estimation to include entire object categories, which requires dealing with changes in appearance due to different poses as well as variations within the category [12] [13]. In order to tackle this problem, some researchers [14] [11] have used a framework called discriminative part-based models (DPMs) within an object detection framework. This approach considers each mixture component to represent a different azimuth section and solves the problem of pose estimation as a structured prediction problem. However, these methods only predict a subset of Euler angles with respect to the object's canonical frame, and do not estimate the object's position or dimensions.

One alternative approach to 3D pose estimation is to make use of existing 3D shape models of objects. This involves sampling hypotheses about the object's viewpoint, position, and size, and then comparing rendered 3D models of the object to the detection window in the image using HOG features [15]. Researchers have also explored using CAD models to estimate object poses [16], but this has only been successful in simpler settings with less challenging detection problems. In more difficult scenarios, such as those with significant occlusion, researchers have developed methods that use dictionaries of 3D voxel patterns learned from CAD models to characterize both the object's shape and common occlusion patterns [3], and refine the object's pose by estimating correspondences between the projected 3D model and image contours. These methods have been evaluated on datasets such as PASCAL3D+ and in tabletop settings with limited clutter or scale variations.

Several recent methods have been developed to detect 3D bounding boxes for driving scenarios, and our method is most closely related to them. Xiang et al. used 3D voxel patterns to cluster the possible object poses into viewpoint-dependent subcategories, which capture shape, viewpoint, and occlusion patterns. They used 3D CAD models to learn the pattern dictionaries and classified the subcategories discriminatively using deep CNNs. Chen et al. tackled the problem by sampling 3D boxes in the physical world and scoring them based on high-level contextual, shape, and category-specific features, assuming the flat ground plane constraint. However, all of these approaches require complicated preprocessing, such as segmentation or 3D shape repositories, and may not be suitable for robots with limited computational resources.

III. Datasets

KITTI is a pervasively used dataset for mobile robots & autonomous driving. It constitutes of several hours' worth of traffic scenarios that were recorded multimodally with sensors like high-definition RGB, grayscale stereo, & 3D laser-cameras. The dataset has image sequences with a resolution of 1242×375, captured at 10 FPS(frames/sec) from a VW wagon traveling in urban environments. The benchmark for 3D object detection consists of 7481 training & 7518 test images with associated point clouds. However, due to absence of labels of KITTI test images we take 80-20% train-test split of train data. The objects in the dataset are annotated with 2D bounding boxes with 6 class labels viz. cars, pedestrians, cyclists, trams, trucks, and vans. The 3D Object detection dataset has been divided into 3 categories, as given in Table I, which were used during evaluation.

TABLE I: KITTI 3D Object detection dataset difficulty levels

Difficulty level	Min. bounding box height	Max. truncation	Max. occlusion level
Easy	40 Px	15%	Fully visible
Moderate	25 Px	30%	Partly occluded
Hard	25 Px	50%	Difficult to see

IV. Methodology

The architecture consists of two main parts: a deep convolutional neural network (CNN) and a geometric reasoning module. The CNN is responsible for feature extraction and produces a feature map that encodes spatial & semantic information about the objects in the image. The geometric reasoning module takes the feature map as input and generates 3D bounding box proposals by combining the CNN output with geometric constraints based on the object's projected 2D bounding box.

A. Feature extraction

1) VGG-19: One of the feature extraction architectures used in this paper is a modified version of the VGG-19 network, as shown in Fig. 1. The VGG-19 network is a widely used convolutional neural network (CNN) architecture for image classification. VGG-19 consists of 19 layers, including 16 convolutional layers and 3 fully connected layers [17]. The convolutional layers are arranged in sequential order, with the input image passing through each layer to extract increasingly complex features. The first 13 layers have 3x3 convolutional filters, while the remaining 6 have 1x1 filters. After every two convolutional layers, the max-pooling procedure is used.

The input to the modified VGG-19 network is a 224x224x3 RGB image, and the output is a feature vector of size 7x7 with 512 channels. This feature vector is then used as input to a separate network to estimate 3D bounding boxes.

2) MobileNet-v2: MobileNet, as shown in Fig. 2, is a deep convolutional neural network architecture designed for efficient computation on mobile and embedded devices. It was first introduced in 2017 by Google researchers Andrew G. Howard, Menglong Zhu, Bo Chen, et al [18]. The MobileNet architecture used in our project is a modified version of the original architecture with an input of 224x224x3 RGB image, and an output of 1280x7x7.

The first layer is a standard convolutional layer with a stride of 2, which reduces the spatial dimensionality of the input feature map. This is followed by a series of depthwise separable convolutional layers, each consisting of a depthwise convolution and a pointwise convolution, with batch normalization and ReLU activation applied after each convolution. The depthwise convolution applies a single filter to each

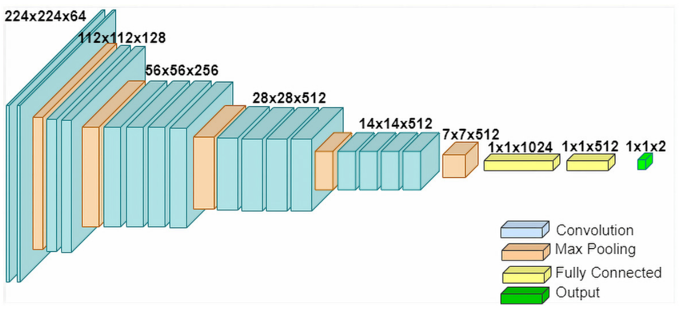

Fig. 1: VGG-19 architecture

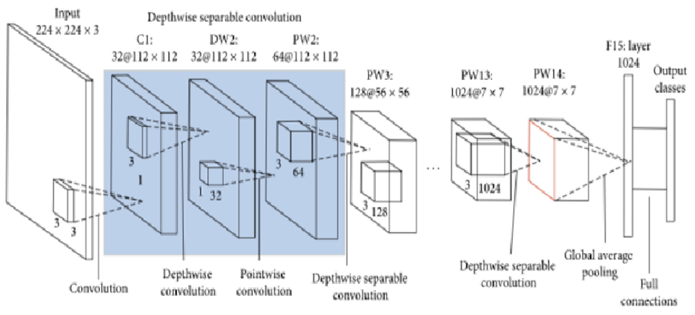

Fig. 2: MobileNet-V2 architecture

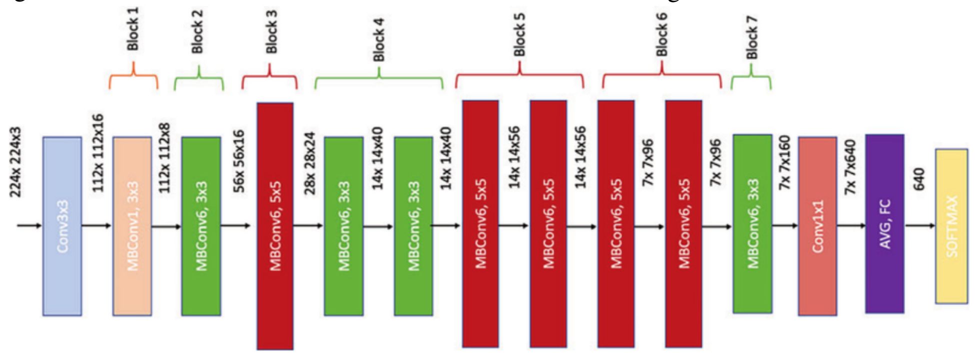

Fig. 3: EfficientNet-V2 architecture

input channel separately, producing a set of output channels with the same spatial dimensions as the input. The pointwise convolution then applies a 1x1 convolution to the output of the depthwise convolution, producing a set of new channels. This operation allows for the creation of non-linear combinations of the output channels from the depthwise convolution, and increases the number of channels.

Even MobileNet architecture employs ”bottleneck” layers, which consist of a 1x1 convolution applied to the input, followed by a depthwise convolution and another 1x1 convolution. This reduces the number of input channels to the depthwise convolution, and the number of output channels from the pointwise convolution, while still allowing for the creation of complex feature representations. Subsequent to depth convolutional layers, the extracted features are sent to the downstream regression layers for 3D bounding box estimation.

Overall, the MobileNet architecture achieves high accuracy on a variety of computer vision tasks while maintaining low computational cost, making it well-suited for deployment on mobile and embedded devices.

3) EfficientNet-v2: The EfficientNet-v2 family consists of three models: EfficientNetv2-S, EfficientNetv2-M, and EfficientNetv2-L. In our project, we have used EfficientNetv2-S, as shown in Fig. 3. It is a smaller and faster version of the original EfficientNetV2 architecture, designed for mobile and edge devices with limited computational resources. The original architecture consists of a stem, multiple stages, and a top [19]. The stem is the initial part of the network that takes the input image and performs some initial feature extraction. The stages are a series of blocks that extract increasingly complex features, and the top is the final part of the network that performs the classification or regression task. The input to the network is a 224x224x3 RGB image with an output of size 512x7x7. The first layer is a stem convolutional layer that processes the input image and reduces its size. The architecture uses a series of convolutional blocks, each consisting of a combination of 1x1, 3x3, and 7x7 convolutional filters with different dilation rates and a squeeze-and-excitation module for channel-wise feature recalibration. The architecture also includes a novel type of block called the ”FusedMBConv”, which combines the 1x1 and 3x3 convolutions into a single operation.

EfficientNet-v2 models use several techniques to reduce computation and memory usage, such as efficient channel gating, weight standardization, and using smaller kernel sizes with higher dilation rates. The architecture also uses compound scaling, where the depth, width, and resolution of the network are scaled together in a balanced way, to achieve better performance while keeping the model efficient.

B. Geometric reasoning

The geometric reasoning module consists of three main components: a 2D bounding box estimator, a 3D bounding box estimator, and a refinement module. The 2D bounding box estimator uses the CNN output to predict the 2D bounding box coordinates of the object in the image. The 3D bounding box estimator takes the 2D bounding box coordinates as input and predicts the 3D bounding box dimensions and orientation. Finally, the refinement module refines the 3D bounding box proposals by minimizing the geometric error between the predicted and ground-truth 3D bounding boxes.

1) 3D Bounding Box Estimation: To make use of the success of 2D object detection in estimating 3D bounding boxes, we rely on the fact that the projection of a 3D bounding box onto a 2D image should fit tightly within its 2D detection window, referenced from Mousavian et al [1]. We assume that the 2D object detector, i.e. YOLO (You-only-look-once), has been trained to produce boxes that correspond to the bounding box of the projected 3D box. The 3D bounding box is defined by its center $T = [t_x, t_y, t_z]$, dimensions $D = [d_x, d_y, d_z]$, and orientation R(θ, φ, ∝), which are determined by the azimuth, elevation, and roll angles. Given the position and orientation of the object in the camera coordinate system $(R, T) \epsilon SE(3)$ and the camera intrinsics matrix K, we can project a 3D point $Xo = [X, Y, Z, 1]^T$ in the object's coordinate system onto the image plane as $x = [x, y, 1]^T$ is:

$$x = K[R\,T]Xo \tag{1}$$

If we assume that the center of the 3D bounding box corresponds to the origin of the object coordinate frame, and we know the dimensions of the object (D), then the coordinates of the 3D bounding box vertices can be described by simple equations, such as $X1 = [d_x/2, d_y/2, d_z/2]^T$, $X2 = [-d_x/2, d_y/2, d_z/2]^T$, and so on, up to $X8 = [-d_x/2, -d_y/2, -d_z/2]^T$. To ensure that the 3D bounding box fits tightly into the 2D detection window, we require that each side of the 2D bounding box is touched by the projection of at least one of the 3D box corners. For instance, if the projection of one 3D corner, $X0 = [d_x/2, -d_y/2, d_z/2]^T$, touches the left side of the 2D bounding box with coordinate xmin, this results in an equation that relates the two, as follows:

$$x_{min} = \left(K[R\,T] \begin{bmatrix} d_x/2 \\ -d_y/2 \\ d_z/2 \end{bmatrix} \right)_x \tag{2}$$

In the above equation, $(.)_x$ refers to the x coordinate of the projected point. We can derive similar equations for the other parameters of the 2D bounding box, such as x_{max}, y_{min}, and y_{max}. Overall, the sides of the 2D bounding box provide four constraints on the 3D bounding box. However, these constraints are not enough to fully determine the nine degrees of freedom (DoF) of the 3D box, which include three for translation, three for rotation, and three for box dimensions. To further constrain the 3D box, we can estimate additional geometric properties from its visual appearance. These properties should be strongly related to the visual appearance and provide additional constraints on the final 3D box.

a) Regression Parameters: The orientation parameters (θ, φ, ∝) have a significant impact on the 3D bounding box. Besides them, we choose to estimate the box dimensions (D) instead of the translation parameters (T). This is because the variance of the dimension estimates is generally smaller and does not change with the object orientation, making it a desirable property when regressing the orientation parameters as well. Moreover, the dimension estimate is strongly related to the appearance of a particular object category and can be accurately recovered if we can classify that category.

b) Correspondence Constraints: To determine the translation of the 3D bounding box, we use the dimensions and orientations that were regressed by the CNN and the 2D detection box. This translation is computed by minimizing the re-projection error with respect to the initial 2D detection box constraints. To find the correct configuration, we consider all possible correspondences between the 2D detection box sides and the 3D bounding box corners, which can result in 4096 configurations. However, assuming that the object is always upright and that the relative object roll is close to zero, we can reduce the number of configurations to 64. Solving for each configuration is computationally fast and can be done in parallel.

2) CNN Regression of 3D Box Parameters:

a) MultiBin Orientation Estimation: Estimating an object's global orientation in camera frame from detection window crop requires crop position within the image plane. In the case of a car moving in a straight line, even though global orientation of the car doesn't change, its local orientation w.r.t. the ray through crop center does change, resulting in appearance changes of the cropped image. The rotation of the car is only parameterized by azimuth (yaw) angle, as seen in Figure 4. Hence, we estimate the local orientation angle of the object w.r.t. the ray passing through the crop center. In Figure 4, the local orientation angle & ray angle change in a way resulting in constant global orientation of the car. During inference, we use the ray direction and estimated local orientation to compute the global orientation of the object.

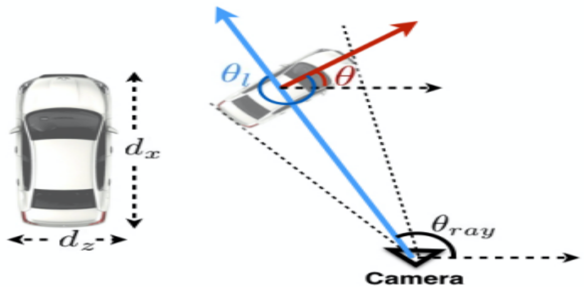

Fig. 4: Left: Car dimensions, ht. of car = dy. Right: Car's Local (θ_l) & global orientation (θ). The local orientation is computed w.r.t. the ray that goes through the crop center. The center ray of the crop is blue arrow. $\theta = \theta_{ray} + \theta_l$. The network is trained to estimate the local orientation θ_l [1]

In our project, we use the MultiBin architecture [1] for orientation estimation. Based on this approach, the orientation angle is divided into n overlapping discrete bins. For each bin, the CNN network estimates a confidence probability (c_i) (that output angle falls within that bin) & residual rotation (to obtain the output angle from orientation of the center ray of that bin). The residual rotation is represented by sine and cosine of the angle, resulting in 3 outputs per bin: (c_i, $\cos(\Delta\theta_i)$, $\sin(\Delta\theta_i)$.

The cosine and sine values are obtained by applying an L2 normalization layer to a 2-dimensional input. The total loss for the MultiBin orientation is thus calculated based on these outputs.

$$L_j = L_{conf} + w \times L_{loc} \tag{3}$$

The confidence loss (L_{conf} is calculated using the softmax loss of all bins. The localization loss (L_{loc}) minimizes the difference between the estimated and ground truth angle in each of the bins that cover the ground truth angle, with adjacent bins having overlapping coverage. The localization loss seeks to minimize the difference between the ground truth and all the bins that cover that value, which is equivalent to maximizing cosine distance, as shown in [1]. The L_{loc} is computed as follows:

$$L_{loc} = \frac{-1}{n_{\theta *}} \times \sum cos(\theta^{*} - c_i - \Delta\theta_i) \tag{4}$$

The equation refers to the number of bins that cover the ground truth angle ($n_{\theta *}$), the center angle of bin i (c_i), and the change in angle that needs to be applied to the center of bin i ($\Delta\theta_i$).

During the inference stage, the bin with the highest confidence value is chosen, and the final output is calculated by applying the estimated $\Delta\theta$ of that bin to the center of that bin. The MultiBin architecture consists of two branches - one for computing the confidences $[(c\_i)]$ and other for computing the cosine and sine of $\Delta\theta$. Therefore, the model needs to estimate 3n parameters for n bins.

In our algorithmic extension, we first replace the head of multibin algorithm [1] with a suitable light-weight extractor then using just the sine & cosine of the orientation angle along with the confidence score, we formulate an equally weighted equation between them as our loss function, instead of using centers or dimensions of bounding box as done in [1]. According to us using orientation angle (sine + cosine), along with confidence score, can be an effective approach.

First, by using just orientation angle with confidence score, the model can focus on learning the essential information needed for 3D bounding box estimation, while ignoring other potentially less relevant information, hence reducing risk of over-fitting, which is especially important in our case of limited training data. Second, when using the raw orientation angle, it is difficult for the model to distinguish between angles that are 360 °apart. By using the sine and cosine, the model can represent the orientation information in a periodic & continuous feature space, hence improving accuracy & prediction stability. Third, the confidence score provides a measure of the model's certainty in its predictions. By using the confidence score as a weight, the model can assign more weight to confident predictions and less weight to uncertain ones. This can help improve the Average Precision (AP) of the 3D bounding box estimation.

On close observation we find that this approach is used in avant-garde 3D object detection frameworks, like [20] and [21], and has been shown to achieve high accuracy on various benchmarks. The dimensions and centre point can eventually be ascertained if the orientation of the bounding box is correct. Regressing for them on top of angle and confidence might increase efficiency in large datasets, but considering the limited dataset & computing resources we proved in the results section that our combination of feature extraction & multibin architecture proves to be more robust (or comparable in some cases) and memory efficient than the framework used in [1].

In the KITTI dataset, for cars and cyclists, the dimension variation is just a few centimeters, so, instead of using a discrete-continuous loss function like the MultiBin loss described earlier, an L2 loss function is used directly. As per the standard practice, for each dimension, the residual value is estimated with respect to the mean parameter value calculated over the training dataset. The loss function for dimension estimation L_{dims} is computed as follows:

$$L_{dims} = \frac{1}{n} \sum (D^{*} - D^{-} - d)^2 \tag{5}$$

where D^{*} represents the ground truth dimensions of the object while D^{-} represents the mean dimensions of objects belonging to the same category as the object being detected. The difference between the estimated dimensions (δ) and the ground truth dimensions is computed and squared, and then scaled by the inverse of the average dimension for the category.

The CNN architecture of our parameter estimation module is shown in Figure 5, the one with a red bounded box. There are five branches in the original architecture (green color), used in [22], where the five branches correspond to computing dimension residual, angle residual, confidence of each bin, viewpoint classification, and the center projection of bottom face (cbf) respectively. In [1], the outer architecture, shown in blue bounded box is used, whereas in our architecture we have only considered using two branches: angle residual, and confidence of each bin, basically regression over the orientation only. All of the branches are derived from the same shared convolutional features and the total loss is the weighted combination of $L = \alpha L_{dims} + L\theta$.

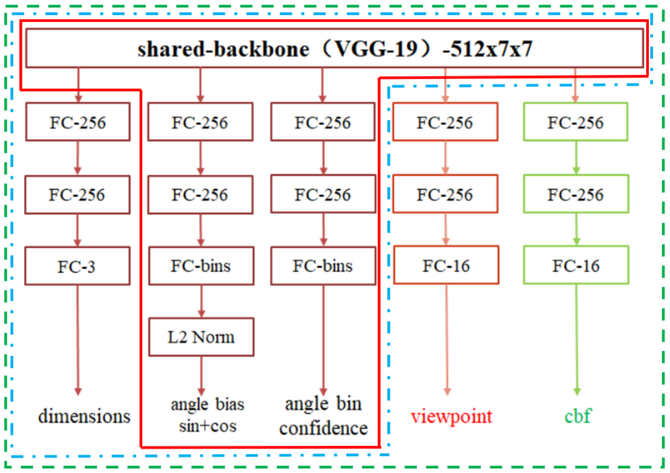

Fig. 5: Architecture for MultiBin estimation for orientation and dimension estimation

TABLE II: For VGG-19: Comparison of the AOS, AP, OS, IOU 3D on official KITTI dataset

Method	Easy				Moderate				Hard			
	AOS(%)	AP(%)	OS	IOU 3D	AOS(%)	AP(%)	OS	IOU 3D	AOS(%)	AP(%)	OS	IOU 3D
Cars (original)	**92.90**	**92.98**	**0.9991**	**-**	**88.75**	**89.04**	**0.9967**	**-**	76.76	77.17	**0.9946**	**-**
Cars (ours)	91.56	91.67	0.998	**0.763**	83.62	85.21	0.981	**0.635**	**76.88**	**78.57**	0.978	**0.625**
Pedestrian (original)	-	-	-	-	-	-	-	-	-	-	-	-
Pedestrian (ours)	**80.63**	**84.62**	**0.953**	**0.764**	**72.58**	**78.57**	**0.923**	**0.633**	**66.21**	**73.33**	**0.904**	**0.624**
Cyclist (original)	69.16	83.94	0.8239	-	59.87	74.16	0.8037	-	52.50	64.84	0.8096	-
Cyclist (ours)	**84.48**	**87.27**	**0.972**	**0.766**	**78.44**	**81.48**	**0.963**	**0.640**	**71.73**	**75.86**	**0.945**	**0.621**

TABLE III: For MobileNet-v2 and EfficientNet-v2: Comparison of the AOS, AP, OS, IOU 3D on official KITTI dataset

Method	Easy				Moderate				Hard			
	AOS(%)	AP(%)	OS	IOU 3D	AOS(%)	AP(%)	OS	IOU 3D	AOS(%)	AP(%)	OS	IOU 3D
Cars (original VGG-19)	92.90	92.98	**0.9991**	-	88.75	89.04	0.9967	-	76.76	77.17	**0.9946**	-
Cars (ours MobileNet)	97.69	97.77	0.999	0.783	89.41	89.80	0.995	**0.722**	82.24	83.02	0.991	**0.642**
Cars (ours EfficientNet)	**98.14**	**98.21**	0.998	**0.788**	**90.03**	**90.16**	**0.998**	0.677	**82.66**	**83.33**	0.992	0.626
Pedestrian (original VGG-19)	-	-	-	-	-	-	-	-	-	-	-	-
Pedestrian (ours MobileNet)	**82.51**	86.27	**0.966**	**0.846**	**75.28**	80.00	**0.941**	**0.641**	**69.38**	74.58	**0.930**	**0.635**
Pedestrian (ours EfficientNet)	81.77	**86.34**	0.947	0.784	74.75	**81.02**	0.923	0.633	68.93	**75.68**	0.912	0.626
Cyclist (original VGG-19)	69.16	83.94	0.8239	-	59.87	74.16	0.8037	-	52.50	64.84	0.8096	-
Cyclist (ours MobileNet)	85.99	86.27	**0.996**	0.763	79.78	81.49	0.979	0.644	73.01	75.86	0.962	**0.641**
Cyclist (ours EfficientNet)	**87.53**	**88.00**	0.995	**0.812**	**81.57**	**83.02**	**0.983**	**0.647**	**75.38**	**77.19**	**0.977**	0.603

TABLE IV: Algorithmic extension for VGG-19, MobileNet-v2 and EfficientNet-v2: Comparison of the AOS, AP, OS, IOU 3D on official KITTI dataset

Method	Easy				Moderate				Hard			
	AOS(%)	AP(%)	OS	IOU 3D	AOS(%)	AP(%)	OS	IOU 3D	AOS(%)	AP(%)	OS	IOU 3D
Cars (ours VGG-19)	98.06	98.21	**0.998**	0.758	89.70	90.16	0.995	0.655	82.38	83.33	**0.989**	0.618
Cars (ours MobileNet)	98.00	98.21	0.997	**0.791**	89.77	90.16	0.996	**0.689**	82.38	83.33	0.989	0.628
Cars (ours EfficientNet)	**98.11**	**98.62**	0.995	0.757	**89.73**	**92.33**	**0.972**	0.652	**82.55**	**85.30**	0.967	**0.644**
Pedestrian (ours VGG-19)	79.93	86.28	0.926	0.778	72.91	80.00	0.911	0.626	66.55	74.58	0.892	0.629
Pedestrian (ours MobileNet)	**81.81**	87.27	**0.937**	**0.803**	**74.59**	81.02	**0.921**	**0.667**	**68.49**	75.58	**0.906**	0.621
Pedestrian (ours EfficientNet)	80.81	**87.17**	0.927	0.778	73.39	**81.56**	0.900	0.628	70.04	**74.58**	0.939	**0.630**
Cyclist (ours VGG-19)	86.22	88.00	0.979	0.759	80.20	83.02	0.966	0.626	72.59	77.19	0.940	0.610
Cyclist (ours MobileNet)	**86.54**	88.55	**0.977**	0.757	81.34	83.92	0.969	**0.658**	73.92	78.17	0.946	0.610
Cyclist (ours EfficientNet)	86.98	**88.10**	0.987	**0.823**	**80.46**	**84.88**	**0.948**	0.633	**74.25**	**79.10**	**0.939**	**0.680**

V. Results

For the evaluation of the different architectures, and algorithms, we use 4 metrics- AOS (Average Orientation Estimation), AP (Average Precision), OS (Orientation Score), and IOU 3D (Intersection-over-union in 3D space); over 3 categories- cars, cyclist, & pedestrian. Even though the original paper doesn't give metrics for pedestrian data, we still chose that because of the clutteredness seen in the data. Also, in the original paper, the author mentions they couldn't achieve good IOU 3D so they have not provided the results for this metric. In Table II, we have presented the results after paper reproduction with the VGG-19 architecture used in [1]. Based on the metrics given in Table II, it can be seen that our implementation yields better or comparable results to the original paper. Secondly, it should be noted, that the original paper's official code wasn't available, and the unofficial implementation of their code also had a lot of bugs, so we had to develop the entire code from scratch.

In Table III, we compare the results after modifying the feature extractor from the original VGG-19 to MobileNet-v2 and EfficientNet-v2. After looking at the metrics, it is empirical that MobileNet-v2, and EfficientNet-v2, both outperform the original VGG-19 architecture on all of the three datasets. In Table IV, we provide the algorithmic extension, using a modified multibin architecture, as discussed in Section IV, for VGG-19, MobileNet-v2, and EfficientNet-v2. Again, based on the Table IV, MobileNet-v2 and EfficientNet-v2, perform better than the VGG-19 architecture, even with the new algorithm.

VI. Conclusion

In this project, we have successfully replicated the results from the original paper [1] in terms of extracting the 3D bounding boxes from a single view for three different dataset categories. We then extended the 3D bounding box accuracies, using the light-weight MobileNet-v2 and EfficientNet-v2 feature extractors, and later a similar performance boost is also observed with our modified multibin architecture. As a result, we conclude that MobileNet-v2 and EfficientNet-v2 outperform VGG-19 architecture in 3D bounding box estimation, from a single view of an RGB image.

References

[1] A. Mousavian, D. Anguelov, J. Flynn, and J. Kosecka, "3d bounding box estimation using deep learning and geometry," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2017, pp. 7074–7082.

[2] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 2012, pp. 3354–3361.

[3] Y. Xiang, W. Choi, Y. Lin, and S. Savarese, "Data-driven 3d voxel patterns for object category recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1903–1911.

[4] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, and R. Urtasun, "Monocular 3d object detection for autonomous driving," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2147–2156.

[5] Z. Cai, Q. Fan, R. S. Feris, and N. Vasconcelos, "A unified multi-scale deep convolutional neural network for fast object detection," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*. Springer, 2016, pp. 354–370.

[6] V. Lepetit, F. Moreno-Noguer, and P. Fua, "Ep n p: An accurate o (n) solution to the p n p problem," *International journal of computer vision*, vol. 81, pp. 155–166, 2009.

[7] V. Ferrari, T. Tuytelaars, and L. Van Gool, "Simultaneous object recognition and segmentation from single or multiple model views," *International journal of computer vision*, vol. 67, no. 2, pp. 159–188, 2006.

[8] F. Rothganger, S. Lazebnik, C. Schmid, and J. Ponce, "3d object modeling and recognition using local affine-invariant image descriptors and multi-view spatial constraints," *International journal of computer vision*, vol. 66, no. 3, pp. 231–259, 2006.

[9] K. Matzen and N. Snavely, "Nyc3dcars: A dataset of 3d vehicles in geographic context," in *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 761–768.

[10] Y. Xiang, W. Kim, W. Chen, J. Ji, C. Choy, H. Su, R. Mottaghi, L. Guibas, and S. Savarese, "Objectnet3d: A large scale database for 3d object recognition," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VIII 14*. Springer, 2016, pp. 160–176.

[11] Y. Xiang, R. Mottaghi, and S. Savarese, "Beyond pascal: A benchmark for 3d object detection in the wild," in *IEEE winter conference on applications of computer vision*. IEEE, 2014, pp. 75–82.

[12] A. Kar, S. Tulsiani, J. Carreira, and J. Malik, "Category-specific object reconstruction from a single image," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1966–1974.

[13] B. Pepik, M. Stark, P. Gehler, T. Ritschel, and B. Schiele, "3d object class detection in the wild," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2015, pp. 1–10.

[14] B. Pepik, M. Stark, P. Gehler, and B. Schiele, "Teaching 3d geometry to deformable part models," in *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 2012, pp. 3362–3369.

[15] R. Mottaghi, Y. Xiang, and S. Savarese, "A coarse-to-fine model for 3d pose estimation and sub-category recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 418–426.

[16] M. Zhu, K. G. Derpanis, Y. Yang, S. Brahmbhatt, M. Zhang, C. Phillips, M. Lecce, and K. Daniilidis, "Single image 3d object detection and pose estimation for grasping," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 3936–3943.

[17] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[18] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[19] M. Tan and Q. Le, "Efficientnetv2: Smaller models and faster training," in *International conference on machine learning*. PMLR, 2021, pp. 10 096–10 106.

[20] Q. Luo, H. Ma, L. Tang, Y. Wang, and R. Xiong, "3d-ssd: Learning hierarchical features from rgb-d images for amodal 3d object detection," *Neurocomputing*, vol. 378, pp. 364–374, 2020.

[21] Z. Shen and C. Nguyen, "Temporal 3d retinanet for fish detection," in *2020 Digital Image Computing: Techniques and Applications (DICTA)*. IEEE, 2020, pp. 1–5.

[22] J. Fang, L. Zhou, and G. Liu, "3d bounding box estimation for autonomous vehicles by cascaded geometric constraints and depurated 2d detections using 3d results," *arXiv preprint arXiv:1909.01867*, 2019.