ENPM809K Final Project
Monocular Pose Estimation of cars using transfer learning and U-net architecture

Hrushikesh Budhale
University of Maryland, College Park
118284217

Nishad Kulkarni
University of Maryland, College Park
117555431

Aniket Paralikar
University of Maryland, College Park
118349343

Abstract

Pose Estimation is crucial to autonomous driving algorithms as the decisions taken by an autonomous car are directly dependent on the objects in it's environment. The dynamic objects in this set are almost entirely comprised of other vehicles, whose next step needs to be predicted to estimate the best possible action a car can take to ensure smooth operation. We implemented a solution based on the EfficientNet method to detect car poses.

1. Introduction

Autonomous driving is currently one of the hottest fields of research. This also means that there are a lot of problems that are currently being researched on for possible solutions. Ideally, it may be expected that a car can drive autonomously using only 2D images or videos as input source for driving decisions. This is however easier said than done as a degree of depth perception is required for making decisions when maneuvering in a 3D world. The predicted relative pose of other vehicles is crucial as it affects the driving decisions of the said autonomous vehicle as some assumptions may thus be made about the detected vehicles which further informs the system about the directions in which the vehicles may be headed.

To find solutions for this problem a competition was hosted by Baidu's Robotics and Autonomous Driving Lab(RAL) along with Peking University to develop an algorithm to estimate the absolute pose of a vehicle (6 DoF) from a single image in a real-world traffic environment. For this, we are provided with an image dataset paired with pose information in csv format that contains the car model type, yaw, pitch, roll, x, y, and z.

In this project we implement and trained a supervised deep learning architecture for detecting the 6 degree of freedom pose for most of the vehicles in a camera frame using only a single image as an input. The model is formed using a pre-trained EfficientNet for feature detection combined with U-net architecture in the latent layer. We achieved an IoU score over 66%. We also perform an ablation study of the network with different feature detector networks and try to find the optimal hyper parameters to increase the important evaluation metrics like Precision, Accuracy, ROC curve and mean position error. Finally we discuss these limitations and possible future work for using this approach to further improve the results.

2. Related Work

A quick look at papers with code pops up with a decent number of papers that try to solve the vehicle pose estimation problem. Here we try to look at a few of them and separate them based on approach.

2.1. 3D constraints based methods

In these types of papers, the authors apply 3D constraints and generalize over objects to estimate depth. Using these constraints, they find a relation between the 2D bounding box of our object classifier and thus finally estimate the vehicle pose[5, 6, 7]. The major difference lies between their approach of these geometric constraints. In addition to these, a depth network may be used to give an idea about the depth of detected objects[8].

2.2. Motion based methods

For motion based methods, the kinematics of the objects in focus are leveraged to estimate the pose of the detected

objects[9].

2.3. Independent object detection

By sub-categorizing objects, it may be more efficient to independently formulate object depth[**?**].

2.4. Directly creating 3D bounding boxes on objects

A CNN can be trained to directly create a 3D bounding box on objects to estimate their projection on 2D image and also the pose in a 3D world.

2.5. Shape reconstruction

Some of the methods propose reconstruction of shape to estimate depth[12]. Of all these methods, the one that seemed to give the best results was to use 3D constraints as it is more robust than other methods and has consistently provided the best results. In our work, we use the methods proposed in the EfficientNet paper[3].

3. Dataset

Figure 1: Sample image from dataset

For solving the our problem, we used the open-sourced dataset Apollo3DCar which is a large scale fully-annotated 3D car database. This dataset has 5277 driving images and over 60,000 Car instances in which each car is annotated with industry grade CAD models.The dataset consists of photos of streets, taken from the roof of a car. Our main challenge is to predict the position and orientation of all the cars in the test images. Every image in the dataset is 3 channel RGB image with, 3384x2710 pixels in width and height. The dataset has images of cars and their pose information. This pose information has following attributes: model type,

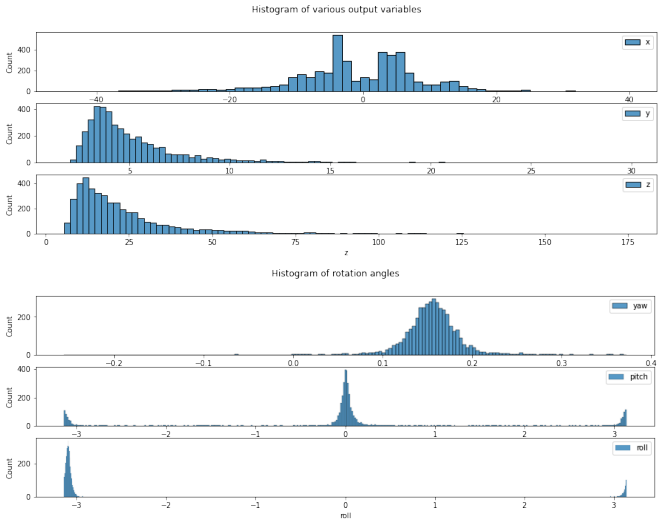


Figure 2: distribution of cars poses in the dataset along x, y, z and roll, pitch, yaw axis

yaw, pitch, roll, x, y and z relative to the camera. In addition, some cars available in the datasets are not of interest, hence we have a dataset consisting of image masks to remove them. Furthermore, we also have information of 3D models of the cars available in .pkl format to compare against the cars in the images. However in our approach we show that it is possible to get good accuracy for both detection and pose estimation of cars without mask images or the car models files while training.

3.1. Data analysis

While analyzing the data, we first looked at the distribution of data labels in all images. This was done by plotting histogram of x, y, z and roll, pitch, yaw of all the cars in the labels of dataset. Using the provided camera matrix, we projected these 3d location in the image frame. After projecting these points on the exact locations of cars in the images 2 important observations were made.

1. Many positions mentioned in the labels are out of frame in left and right direction. We handled this case by extending the image shape horizontally.

2. Most of these points were lying in the same plane. This observation is logically true because most cars are on flat road surface. We used this information to optimize the predicted depth of the car by fitting it's position using linear regression.

3.2. Pre-Processing

Data pre-processing is a major step in this project. First we are separating the 4252 image dataset into training, and

Figure 3: Generated detection mask for the respective image

Figure 4: Image after pre-processing

validation sets with 98 and 2 percent split. Then, we split this pre-processing into 2 parts one for image and another for labels.

3.2.1 Processing Labels

1. Extract pose: Labels for all detected car poses are given in single CSV row. To extract the meaningful information out of it, we first split the row in multiple rows with each row containing 1 detection and then convert each reading into float values, These values are then being used in the following Image pre-processing step.

2. Create training Mask: The resulting pre-processed inputs are 7 channel tensors with first channel representing the car position by setting binary value of 1 where car is present and zero elsewhere. Then later 6 channels representing the pose of a vehicle. The processed tensor takes final shape of 7 x 40 x 128 values.

3.2.2 Processing Image

We are first resizing image, such that it can be passed to the network. Then we divide the pixel values by 255 to bring all pixel values within 0 to 1. Since all images have the cars in lower half, we are cropping the images and only the lower half of it is being used. In the EDA process we found that few of the coordinates mentioned are going out of given frame's width. To compensate for this we are extending the width of the image by padding left and right sides of the images by the nearest columns in the images. Finally we resize the image to lower resolution to lower resolution .

3.2.3 Visualization

To be able to evaluate the correctness of output it is important to visualize it. For our project it was a challenging part to visualize these poses of all cars and comment on their accuracy. To solve this problem we used the approach of creating a bounding box of with the size of standard car's size footprint. Using the help of camera matrix and poses

Figure 5: projection example

extracted from the corresponding images, these bounding boxes were projected in front of the camera. By adjusting the camera's height and orientations parameters. We were able to visualize the label pose for every car in the frame. This visualization was also important for validating the correctness and evaluating the performance of our model output. The output of this visualization method can be seen in the Figure 5.

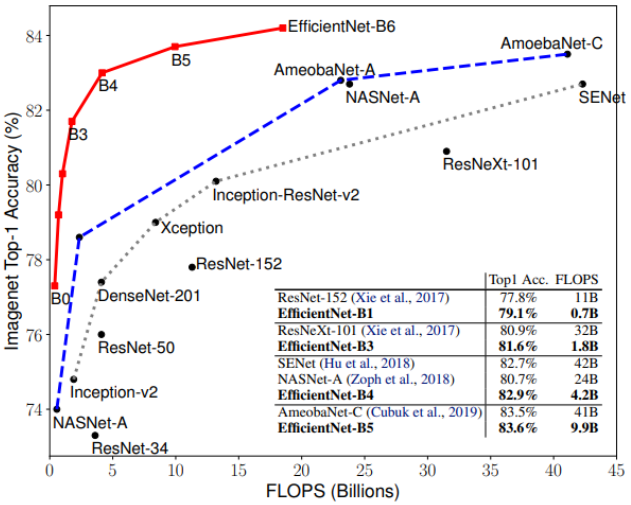
4. Method

To approach this problem we decided to first use a part of the network to detect and then segment that car. After segmenting, apply 2D convolutional layers to extract the 6 channel pose out of it.

To approach this problem, initially we thought of training the entire network ourselves. However, this would have required us to train a larger model with first part of network to detect the instances of cars and next part to generate masks for these detection and then subsequent layers to extract pose out of it. To alleviate the training process of such huge network, we decided to use transfer learning approach where we will use pre-trained model for feature extraction and on it's output add our network to segment the cars and compute their pose.

4.1. Object Detection

Since detection of cars is one of the crucial and heavily researched problem in the computer vision, there exist multiple pre-trained networks which have shown high level of accuracy in for car detection tasks. To select the best network for our project we focused on 2 metrics namely



	Top1 Acc.	FLOPS
ResNet-152 (Xie et al., 2017)	77.8%	11B
EfficientNet-B1	**79.1%**	**0.7B**
ResNeXt-101 (Xie et al., 2017)	80.9%	32B
EfficientNet-B3	**81.6%**	**1.8B**
SENet (Hu et al., 2018)	82.7%	42B
NASNet-A (Zoph et al., 2018)	80.7%	24B
EfficientNet-B4	**82.9%**	**4.2B**
AmeobaNet-C (Cubuk et al., 2019)	83.5%	41B
EfficientNet-B5	**83.6%**	**9.9B**

Figure 6: EfficientNet performance showing FLOPS vs. Accuracy on Imagenet

accuracy and inference speed. Since we were planning of training a connected network on the extracted features from the pre-trained backbone, inference speed was important factor to reduce the experimentation time and iterating over our network architecture in limited time.

Based on the study conducted between highly used networks such as ResNet, InceptionNet, SeNet, MobileNet-V2 and EfficientNet, it has been found out that Efficient net has specially been designed to achieve the highest accuracy with minimum number of network parameters which in tern reduces the inference time significantly. 6 On imagenet dataset, the EfficientNet-B1 achieveed 78.8% accuracy with each image taking only 0.098s for inference which is 5.7x speedup over inference speed of ResNet-152.

4.2. Image Segmentation

In the segmentation approach we use the original image and extract its features using using encoder model and then by up-sampling the output of those features using the decoder model it generates a mask for specific class. This approach of using encoder decoder model has shown high accuracy for many segmentation applications.

4.2.1 U-net Model

In this approach U-net is one of the common architecture, where output from previous encoder layers is combined with the decoder outputs to make use of the high quality features from the input and generate more coherent output masks. Hence for our project we implemented the U-net architecture to train the model to calculate the car's pose.

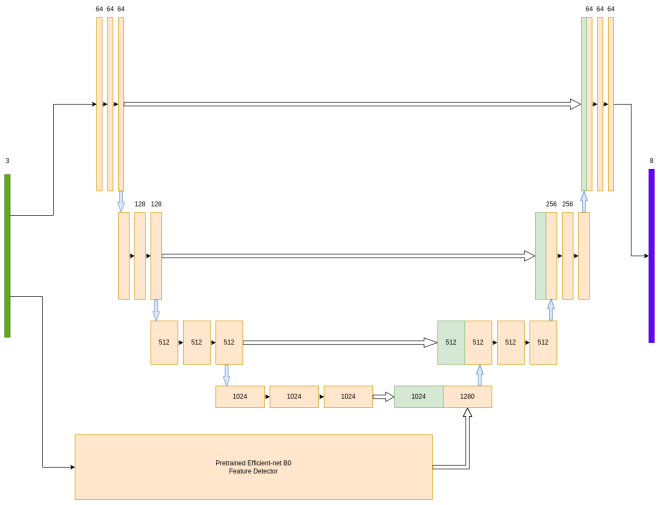


Figure 7: Model architecture

4.3. Combining model outputs

Now to utilize the feature output from the pre-trained model while also making use of the U-net architecture, we passed the same image as a input to both EfficientNet and U-net models. Then the output of from the EfficientNet was appended in the latent space output of the encoder model. This combined output was then used in the decoder network. Since output from the EfficientNet could detect and localize the cars and then the output from encoder could extract the shape of that car, we expected the decoder to be able to decode the 6 channel pose from this input.

5. Implementation Details

In this section we describe the details about our implementation of architecture as shown in Figure 7.

5.1. Encoder Network

In the encoder architecture, we took the input of 3 channel RGB image which was passed through the convolutional layer with kernel size of 3x3, padding and stride of 1 pixel. This was followed by a batch normalization layer and it's output is then processed through ReLU activation layer. After two iterations of these operations the output is passed through the 2d max-pooling layer with stride size of 2. giving us the output tensor with half the size in width and height.

These operation of convolution, batch normalization, ReLU and MaxPooling are then consecutively applied 4 times to get the final encoded feature tensor with 1024 number of channels. The output from each of these 4 operations is also stored to be used later in the decoder network.

Table 1: Layers in encoder network

Layer	in channels	out channels
2x Conv+BN+ReLU	3	64
MaxPool	64	64
2x Conv+BN+ReLU	64	128
MaxPool	128	128
2x Conv+BN+ReLU	128	512
MaxPool	512	512
2x Conv+BN+ReLU	512	1024
MaxPool	1024	1024

5.2. Decoder Network

The input to the decoder network is a combined tensor formed by the output from encoder and EfficientNet along channel dimension. Since the Encoder and EfficientNet outputs have 1024 and 1280 channels respectively the combined input of 2304 channels is passed to the decoder network.

In the decoder network the output is upsampled using 'bilinear' interpolation method and then passed through the 2 passes of convolution, batchnorm and relu activations as mentioned in the encoder section.

5.3. Loss Function

At the end of the decoder we get the output with the same size at the original input image with 8 channels. Here 1 extra channel represents the confidence or probability of the car at that pixel location. This output is then split into 2 parts. In the first part the error in the first channel is calculated using one of the classification criterion called binary cross-entropy also called as BCECriterion, which is applied on the sigmoid output of the detections. In the second part, we apply Regression criterion called as AbsCriterion which is calculated by taking the mean absolute value of the element-wise difference between inputs. Both the losses are then added to calculate the total loss.

Table 2: Layers in decoder network

Layer	in channels	out channels
Upsample	2304	2304
2x Conv+BN+ReLU	2304	512
Upsample	1024	1024
2x Conv+BN+ReLU	1024	256
Upsample	512	64
2x Conv+BN+ReLU	128	8

6. Experiments

6.1. Training

For training, we used the Nvidia RTX 3080, 8GB GPU. With this configuration we were able to load network with batch size 4. In the training process we passed 98% of the training dataset for training and remaining 2% for validation.

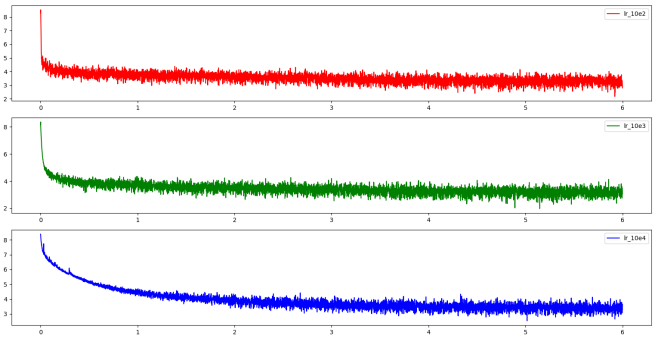


Figure 8: training losses

Table 3: Hyper-Parameters used while training the model

Parameter	Value
Learning Rate	10e-4
Optimizer	Adam
Criterion	BCECriterion + AbsCriterion
Learning rate scheduler	StepLR
Gamma (gamma)	0.1
Batch Size	4

We trained the network using SGD with Adam optimizer. Based on different learning rates the model trained with learning rate of 10e-3 showed high accuracy while keeping the training time lower. We performed the training for 6 epochs. To reduce the leaning rate after iterations we used the StepLR learning rate scheduler with gamma value of 0.1. During this training we found out that loss start at around 82 and gradually decreases to 26 within first 3 epochs and after it. it remains somewhat steady around that range.

We ensured that the network is not over-fitting on the dataset by visualizing the validation losses. As we can see in the Figure 9, the validation loss stopped decreasing after nearly 5 epochs.

6.2. With Different learning rates

To ensure that we train the network properly we tried training the network with 3 different learning rates ranging from 10e-2 to 10-4. As we can see from the Figure 8 the training with 10e-2 shows higher loss, where at training with 10-4 takes 5 epochs to reach the same loss level as 10e-3.

6.3. With and without freezing weights

For the feature detection part we used the pre-trained EfficientNet-B0 model with last layer removed. And its

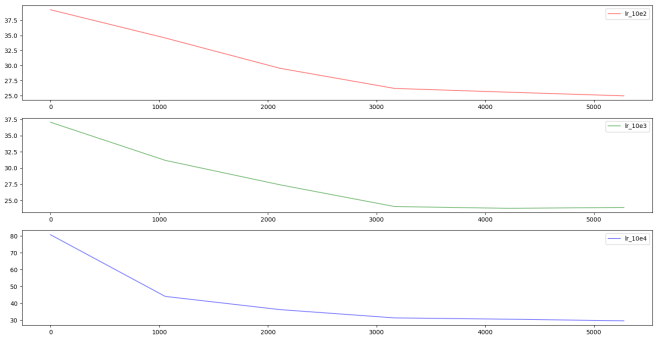


Figure 9: validation losses

output combined with the U-Net model. While experimenting we found out that by freezing the weights of pre-trained efficient-net our loss was reduced by more than 50%. We believe that this happens due to the fact that, pre-trained model has already fine-tuned kernels. While doing back propagation these weights gets disturbed leading to higher loss.

6.4. With different version of backbone

EfficientNet has been developed with 6 different versions ranging from B0 to B6. Hence we tried training our network with 2 different versions namely B0 and B4. In this experiment we observed that the models loss doesn't show any improvement with the updated B4 version of the model. However Number parameters in other words network size increases significantly resulting in slower training process.

6.5. Evaluating the model

For evaluation we use 3 different metrics. For measuring the performance accurate detection of cars we use the Intersection over Union metric which is also called as Jaccard score. This is calculated by counting the number of correct predictions divided by the sum of total predictions and total expected results. With our network we achieved the IoU score of 67.34% at the prediction threshold of 0.1. One example of this thresholded prediction can be seen in the Figure 10.

We also calculated the ROC curve for our trained model to analyze the behavior of the model with increasing threshold value for the detections of cars. This was calculated by first computing the True positive and False Positive rates at each threshold value. As we can see in the Figure 11. The model achieves more than 0.8 TPR while keeping the FPR below 0.1

Finally we also calculate the relative error between the detected cars positions, by first selecting the nearest car to the predicted car position and calculate it's root mean squared error for all car in the image for all the images in the test set. With this metrics we found out that most of the

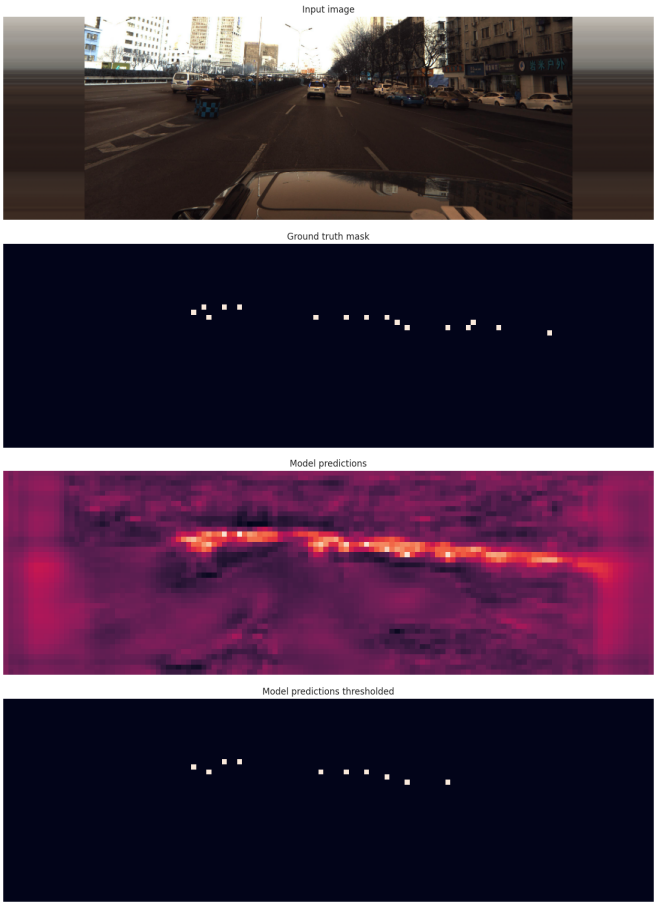


Figure 10: inference input

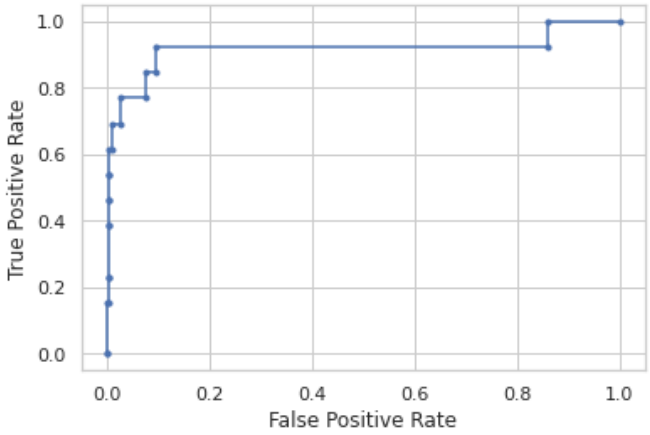


Figure 11: roc curve 1

detected cars are within 4.5 meter distance from the actual cars which achieves the precision score of 26.29%.

1. Each row of the label in CSV contains 6 parameters like position x, y, z and orientation roll, pitch and yaw

Figure 12: Final result

of the detected vehicles along with their ids.

2. We found that poses of the cars in the dataset follow normal standard distribution with pitch and roll centered at zero and yaw centered around 1.8 radian but has wider distribution since cars in the scene can be going in any direction. Positional data of all cars follow log normal distribution in y and z, with more number of cars near 10 meter distance and then gradually falling till 100 meters.

3. When plotted the histogram of number of detected in each cars, we found that most of the images contain 8 to 10 instances of cars.

4. Since cars are on the flat road surface, we expected to have flat distribution of 3d positions for all the detection, plotting this in scatter plot, revealed the same results.

7. Limitations

1. In the case of EfficientNets, we have a neural network architecture that has significantly less compute and more data movement than comparable networks. As a result, EfficientNet does not yield satisfactory performance on hardware accelerators.

2. It does not have any support for low level libraries like cuDNN and OneDNN for achieving optimized performance on any underlying hardware

3. In our approach, we haven't used the mask for excluding the detection of cars by incorporating that data, the accuracy of model can further be improved. Our approach also fails to detect the cars which are partially out of the frame however the dataset labels expect them to be detected. This can be improved by trying better feature detector which can detect occluded cars.

8. Conclusion

We were successfully able to train and implement a deep-learning model consisting of an EfficientNet as a backbone combined with U-net neural network architecture to determine the 6-degrees of freedom pose for most of the vehicles for our images in the given data set. And studied of the network with different feature detector networks and tried to find the optimal parameters to increase Precision and reduce the mean position error.

In addition, to further optimize the results of the application we can explore the capabilities of EfficientNetV2 as a pre-trained layer of our network to perform feature extraction. We can also use a mask to detect images of cars that are partially out of frame.

References

[1] H. Tang, Y. Wang, W. Yuan and Y. Sun, "An Efficientnet Based Method for Autonomous Vehicles Trajectory Prediction," 2021 IEEE International Conference on Computer Science, Electronic Information Engineering and Intelligent Control Technology (CEI), 2021, pp. 18-21, doi: 10.1109/CEI52496.2021.9574480.

[2] L. -A. Tran and M. -H. Le, "Robust U-Net-based Road Lane Markings Detection for Autonomous Driving," 2019 International Conference on System Science and Engineering (ICSSE), 2019, pp. 62-66, doi: 10.1109/ICSSE.2019.8823532.

[3] Mingxing Tan, Quoc V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks" 2019 International Conference on Machine Learning.

[4] Xingyi Zhou, Dequan Wang, Philipp Krähenbühl, "Objects as Points" arXiv:1904.07850.

[5] Arsalan Mousavian, Dragomir Anguelov, John Flynn, Jana Kosecka, " 3D Bounding Box Estimation Using Deep Learning and Geometry".

[6] Garrick Brazil, Xiaoming Liu, "M3D-RPN: Monocular 3D Region Proposal Network for Object Detection"

[7] Mingyu Ding, Yuqi Huo, Hongwei Yi, Zhe Wang, Jianping Shi, Zhiwu Lu, Ping Luo, "Learning Depth-Guided Convolutions for Monocular 3D Object Detection"

[8] Peixuan Li, Huaici Zhao, PengFei Liu, Feidao Cao, "RTM3D: Real-time Monocular 3D Detection from Object Keypoints for Autonomous Driving"

[9] Garrick Brazil, Gerard Pons-Moll, Xiaoming Liu, Bernt Schiele, "Kinematic 3D Object Detection in Monocular Video"

[10] Yu Xiang, Wongun Choi, Yuanqing Lin, Silvio Savarese, "Subcategory-aware Convolutional Neural Networks for Object Proposals and Detection"

[11] Jakub Sochor, Jakub Špaňhel, Adam Herout,"BoxCars: Improving Fine-Grained Recognition of Vehicles using 3-D Bounding Boxes in Traffic Surveillance "

[12] Jason Ku, Alex D. Pon, Steven L. Waslander,"Monocular 3D Object Detection Leveraging Accurate Proposals and Shape Reconstruction"