Final Project Report

Name: Chaofu Zhang, Qianhui Cen, Yuan Le

Introduction

The goal of this project is to anticipate how a vehicle would do autonomous driving based on the analysis of the surroundings (position of nearby automobiles). We choose this problem because we want to explore the truth of autonomous driving and find out if there is space to improve self-driving to make this technique widespread. In this report, we will introduce our dataset, talk about how we pre-processing our data and choose parameters, discuss our deep learning network and training algorithm, and do the analysis based on the results.

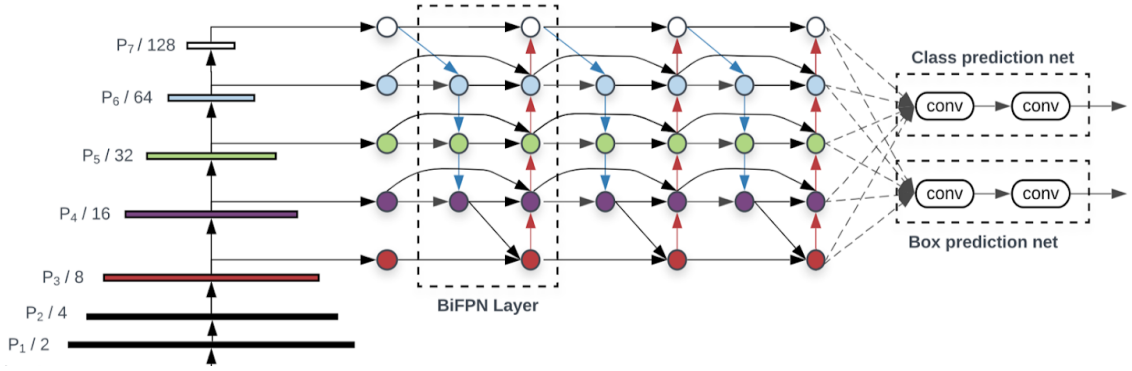
Description of the data set

For this project, we choose Peking University's autonomous driving data from Kaggle. This dataset contains street photos taken from a car-equipped camera and the related labeling information of cars' pose. The 6283 images are divided into 4220 for the train set and 2063 for the test set. The car-related pose information is formatted into the string type: model type, yaw, pitch, roll, x, y, z. In addition, the image masks data allows us to remove useless car information(cars which are too far away are not in our interest). The dataset has over 60,000 labeled 3D car instances from real-world images based on a series of industry-grade car models. It is used for the comparison with the cars in the images and rotation reference. Lastly, the data of the camera position is utilized to ensure the position of each car in the images, which means the basepoint for our pose information, is the position of the camera.

Data Processing and Experimental setup

In the pre-processing part, since the original image size is large, to save the computation power, we need to resize them to a proper size to fit in our model input. After the EDA part, we've found all vehicle points (draw based on x, y, z) gathered at the bottom half of the image, yet some of the center points are outside of the image boundary. To get rid of these issues, we cut the top half image off and just keep the bottom half as our input images. In addition, we also need to expand the image width to include outlier points. After that, we resize the pictures to 2048*640 and do the standardization to keep pictures all in the same size. While resizing the image, the pose information would change if the weight-height ratio is changed, therefore, to avoid this happen, we kept the ratio of width and height of the image unchanged. The original ratio is 1:2.5, while our resized image has a ratio of 1:3.2, so we applied two paddings to widen the images. The padding size for the left part is 1/8 of the images and for the right part is to fill up to 2048. Furthermore, we randomly flipped 10% of the images vertically as the augmentation, and divide the image RGB values by 255 to standardize the input. During the training part, we use minibatch with a batch-size of 4 and set the epoch number to be 10. We use Adam as our optimizer and Learning Rate as 0.001. The combination of Binary Cross-Entropy for classification loss and L1 for regression lose was set as our final performance index.

Description of the deep learning network and training algorithm



The deep learning network used in the project is a variation version of the EfficientDet. In this network, an EfficientNet network is used as the backbone network for the feature extraction on various scalings. The core idea of the EfficientNet is the compound scaling of the network in multiple dimensions (input resolution, depth, width). Instead of scaling only one or two dimensions of the network, EfficientNet expands the network in all dimensions. Compared to the old detection networks, EfficientDet achieves good performance and reduces the number of parameters. After extracting feature maps from multiple scales, a fusion network module called BiFPN is adopted to efficiently combine the features in different scales together for further prediction and classification. The overall architecture of the EfficientDet can be found in the above figure.

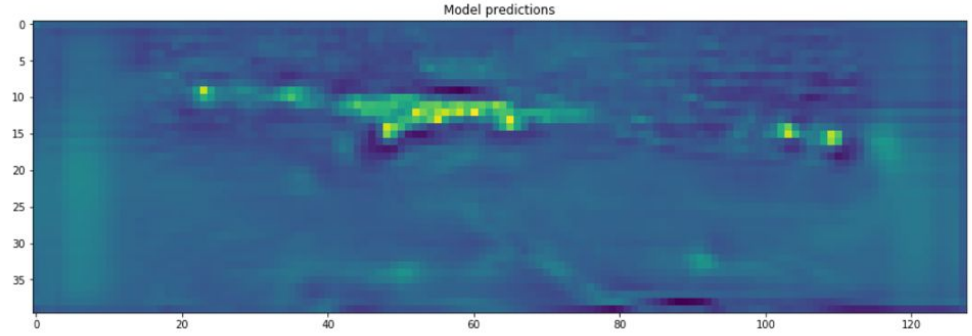
Since EfficientDet is a newly published work, we did not find a solid implementation or an official implementation available online. Alternatively, we found a PyTorch version implementation of the EfficientNet from GitHub, so we implemented the BiFPN network module based on our understanding, and build the EfficientDet network based on the EfficientNet and our implemented BiFPN modules.

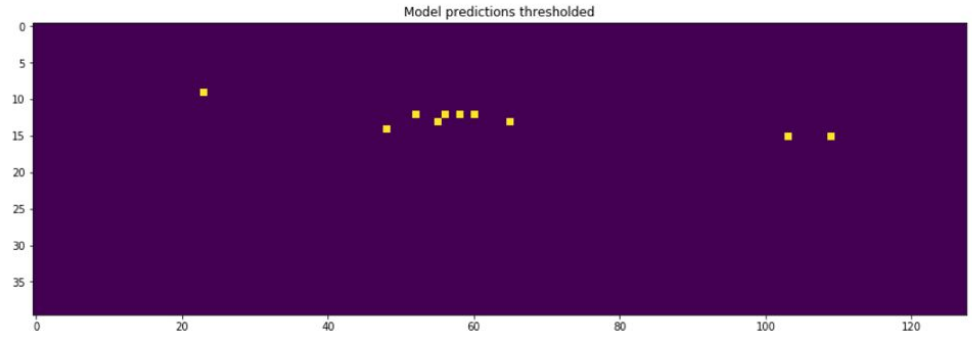
The original EfficientDet aiming at predicting bounding boxes for object detection, while our target is a 6-DoF pose, so we cannot train or predict based on the bounding box. We then inspired by a work called CenterNet, which aims at predicting the center point heatmap of objects, and regress the bounding box features. In this project, we can predict the heatmap and then regress the 6-DoF pose features in a similar way. Also, the original EfficientNet package only provides feature_extract function for the last layer, so we sub-classed a customized EfficientNet class from the original one and build our network based on our expanded class. We also implemented our customized prediction head layers to suit for our targets. In general, our predictor has two heads, one for center-point classification and the other for 6DoF pose regression. The predictor takes the fused multi-scaled feature maps from the BiFPN network as inputs and makes predictions with respect to each scale. The results are then upsampled to the same size and stacked together, and are fed to the output layer to provide our final prediction of the mask classification heatmap and the 6-DoF pose. Due to the limitation of computational power and time constraint, we only implemented the EfficientDet-D0 model, which uses EfficientNet-B0 as the backbone, and has 2 BiFPN modules with 64 channels in the fusion network.

Results

After the 10 epochs training, the training loss for our model is more than 20, which is really high. The main reason for the high loss comes from the BCE(Binary Cross Entropy) of the center point classification. Without BCE, the loss of the regression part is only about 0.4. The first figure below shows the heatmap with the prediction targets, it looks good but not very clearly. After doing the threshold operation, each

prediction dot can be discovered distinctly in the second figure. By ensuring the position of each car in one image, the prediction of autonomous driving can be done by avoiding existing cars in sight.





Summary and conclusions

In conclusion, we implemented the EfficientDet as our network and performed prediction on the 6-DoF pose from camera captured pictures. We borrowed the idea of predicting center points heatmap and regressing on the center point related features from the CenterNet, and we combined the predictions from multiple scaled feature maps similar to U-Net, as we upsampled the outputs to the same size, stacked them together, and further fed the stacked outputs to a final output layer to predict the center point heatmap and pose features.

Due to the time and computational power limitations, our performance is not comparable to others in the Kaggle leaderboard. In our future work, we may further tune the network and modify the network to improve performance. We may also consider advanced data augmentation methods or including dropout layers in our networks to prevent overfitting.

References

Dataset retrieved from: https://www.kaggle.com/c/pku-autonomous-driving/overview

Tan, M., Pang, R., & Le, Q. V. (2019). EfficientDet: Scalable and Efficient Object Detection. *arXiv preprint arXiv:1911.09070*.

Mingxing Tan, Quoc V. Le.(2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *arXiv preprint arXiv:1905.11946.*

Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, Serge

Belongie.(2019). Feature Pyramid Networks for Object Detection.*arXiv:1612.03144*

Xingyi Zhou, Dequan Wang, Philipp Krähenbühl.(2019) Objects as Points.

arXiv:1904.07850