Scene Understanding for Autonomous Vehicles

Roger Marí [1], Joan Sintes [2], Àlex Palomo [3], Àlex Vicente [4]
Universitat Pompeu Fabra, Image Processing and Computer Vision Group
{ [1]roger.mari01, [2]joan.sintes01, [3]alex.palomo01, [4]alex.vicente01 }@estudiant.upf.edu

***Abstract*—This work presents a series of experiments related to scene understanding for autonomous vehicles. Different deep learning architectures are employed to address three main and increasingly complex Computer Vision problems. VGG and ResNet are tested for object recognition; YOLOv2 and Faster R-CNN for object detection; and FCN8, SegNet and U-Net for semantic segmentation. Several experiments were carried to exploit the possibilities of the implemented networks, including learning transfer, data augmentation, parameter tuning and structure modifications. The presented results are extensively discussed and neatly show the potential of deep learning for the proposed tasks.**

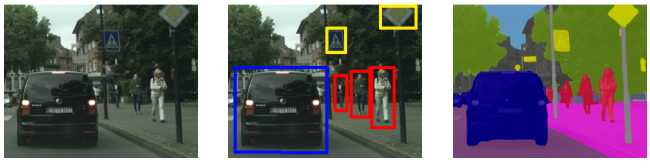
***Index Terms*—deep learning, autonomous driving, object recognition, object detection, semantic segmentation.**

I. Introduction

Autonomous vehicles are expected to bring increased safety, greater efficiency and lower energy consumption to the smart cities of the future. They also represent a unique opportunity to improve the quality of life of disabled individuals, to reduce traffic congestion and to optimize the use of parking spaces. As a result, autonomous driving has become one of the most active fields of research in Computer Vision in the recent years.

We present a series of experiments related to object recognition, object detection and semantic segmentation for autonomous driving. Each of these tasks produces increasingly more complex results towards a common end: understand a traffic scene based on the objects it is composed of.

Different deep neural networks and image datasets are employed to address the aforementioned problems. It should be noted, however, that our work focuses on analyzing and comparing a broad range of architectures instead of obtaining a single top performing model. We also seek to improve the performance of the baseline models by means of fine-tunning, hyper-parameter optimization, regularization techniques or structure modifications. The open-source code is available at `https://github.com/rogermm14/mcv-m5`.



(1) (2) (3)

Fig. 1: Example of input and output of the proposed system. From left to right: (1) Input image, (2) Object detection and recognition result (window color denotes class, i.e. blue for *car*), (3) Semantic segmentation result.

Fig. 2: Images from the TT100K dataset (first row), the BelgiumTS dataset (second row), and the KITTI dataset (third row).

For object recognition we tested VGG [1] and ResNet [2]. The TSingHua-TenCent 100K (TT100K) dataset [3], the Belgium Traffic Sign (BelgiumTS) dataset [4]) and the KITTI dataset [5] were used for training. For object detection we experimented with YOLO [6] and Faster R-CNN [7]. TT100K [3] and the Udacity dataset [8] were used for training. Finally, for semantic segmentation we compared FCN8 [9], SegNet [10] and U-Net [11]. In this case, the Cambridge-driving Labeled Video Database (CamVid) and the KITTI dataset for image segmentation were used to train the different networks.

II. State of the art review

A. Object recognition

In Computer Vision, the task of automatic object recognition used to be tackled using handcrafted features and classical classifiers (such as SVMs) until the breakthrough of deep neural networks a few years ago. In 2012 AlexNet [12], a convolutional neural network (CNN), was the first deep learning system to outperform the classical approaches, winning the ImageNet Large Scale Visual Recognition Competition (ILSVRC) [13]. From then on, multiple CNN architectures achieving better and better results have been introduced. The VGG model, that came second in the 2014 ILSVRC, was a major contribution to the field and highlighted the influence of depth in classification (using small 3×3 convolutions) [1]. In 2015 the winner of the ILSVRC was the ResNet, which reduced the influence of the vanishing gradient problem by allowing blocks of convolutional layers to fit residual mappings [2]. The ResNet gave room for much deeper architectures, which are able to perform even better than human subjects. Finally, the most recent ILSVRC winner, in 2017, was the Squeeze-and-Excitation (SE) net [14]. SE networks use a new unit, the squeeze-and-excite block, to capture channel-wise global information and use it to scale activations. SE blocks achieve significant performance improvements for existing state of the art deep architectures at low computational cost.

B. Object detection

Object detection in deep learning is a more challenging task than object recognition, as it requires both predicting candidate bounding boxes and their corresponding class probabilities. In 2013, the Region based CNN (R-CNN) [15] was presented, where a region proposal system was combined with a classification model using CNNs that achieved state of the art performance. The original R-CNN extracted around 2000 region proposals from each input image, computed CNN features for each proposal, and then classified each region using linear SVMs. In 2014, with the apparition Fast R-CNN, the method was redesigned as a single-stage training algorithm with shared CNN computations among different region proposals [16]. These innovations provided 9× training speed and 213× test speed, allowing Fast R-CNN to achieve near real-time rates when ignoring time spent on region proposals. The region proposal task, that represented a computational bottleneck, was addressed in 2015 by means of the Faster R-CNN [7]. The Faster R-CNN system achieves a frame rate of 5fps, including all steps, by sharing features between region proposal and detection networks. The YOLO architecture came after as an alternative to achieve 45fps or 155fps with almost the same accuracy [6]. This network divides the image into regions and predicts bounding boxes and probabilities for each region, all together in a single evaluation. Other state of the art architectures for object detection, like the Single Shot multibox Detector (SSD), followed similar strategies to eliminate the need of object proposals and gain speed [17].

The different networks for object detection are widely studied and discussed in the context of autonomous driving. In [18] the importance of strict region overlapping between bounding boxes (instead of the classic 50%) is highlighted and R-CNN results are consequently considered insufficient. As a possible solution, a method that learns to generate class-specific 3D object proposals from monocular views is proposed.

C. Semantic segmentation

We reviewed the success of deep learning in image classification in the recent years (section II-A). Semantic segmentation, or classification at pixel level, is a natural next step of increased difficulty with respect to standard classification.

The first remarkable deep learning models to efficiently face the segmentation problem date from 2015. In [9], a popular classification model (VGG-16) was reused for segmentation by substituting the dense layers by 1×1 convolutions, allowing the input images to take arbitrary sizes, and turning the model into a fully-convolutional network (FCN). The subsampling caused by the pooling layers of the VGG-16 was compensated via transposed convolutions to recover spatial resolution. The FCN models were the first to allow end-to-end, pixels-to-pixels, efficient training via backpropagation. FCNs are typically trained by initialization using image classification weights and a subsequent fine-tuning for segmentation.

Shortly after the apparition of FCN, SegNet was presented as an encoder-decoder architecture also capable of achieving state of the art segmentation [10]. The encoder-decoder architecture consists of a contracting path, that reuses the blocks of a classification net (VGG-16 in the case of SegNet) to infer semantic information or *what*; and a decontracting path that inverts the previous and recovers spatial resolution or *where*. SegNet inverts the pooling operations by producing sparse feature maps with values at the maximum indices, that are stored after each maxpooling. The sparse feature maps are turned to dense feature maps via subsequent convolutions.

In parallel, U-Net appeared as an alternative originally conceived for segmentation of biomedical images [11]. U-Net does not reuse the pooling indices but instead transfers the entire feature maps (at the cost of more memory) from the encoders to the corresponding decoders and concatenates them to upsampled (via deconvolution) decoder feature maps.

Many deep learning architectures for semantic segmentation have been introduced after the aforementioned ones. DeepLab [19] or FCN-ResNet [20] are some other popular examples.

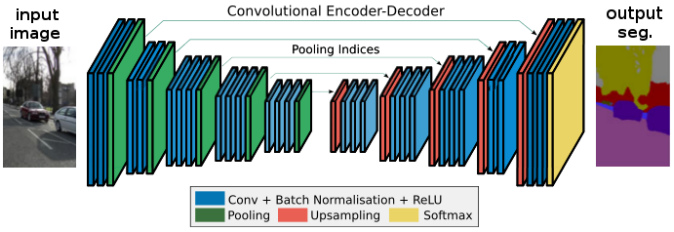


Fig. 3: SegNet-VGG16 architecture for semantic segmentation.

III. Object Recognition

The goal of this module is to recognize the kind of object in a given window (input image). This can be understood as a classification process: the input image is assigned a label from a list of objects of interest (or classes). Pedestrians, cyclists and vehicles (e.g. cars, vans, trucks or motorbikes) are objects that an autonomous car should be able to recognize.

In our initial experiments we trained the VGG-16 from scratch using the TT100K dataset. 45 different classes of traffic signs were considered. Notice that the images from TT100K have 64x64 pixels, but the VGG has a fixed input size of 224×224. In the light of this consideration, we tested two methods to produce images of 224x224 from the TT100K dataset: (1) image resize to 224×224 and (2) resize to 256×256 followed by a random crop of 224×224. In both cases a preprocessing step of mean subtraction and standard deviation normalization (computed using the entire training set) was applied. After training with TT100K, we fine-tuned the fully-connected layers of the VGG-16 for the BelgiumTS dataset to assess how good the learned features generalized to a different context (country) for similar traffic signs.

In a second stage of experiments, we trained the VGG-16 using another dataset with no traffic signs but pedestrians, vehicles or cyclists instead. The KITTI dataset, that considers 8 different classes (*background*, *car*, *van*, *truck*, *pedestrian*, *person sitting*, *cyclist* and *tram*), was employed in this case. We trained the model from scratch and compared the performance to the one achieved by fine-tuning the fully-connected layers using the weights pretrained with ImageNet.

Having gained experience from the VGG-16, we moved to the ResNet-50. Again, we trained the model from scratch and compared the performance to the one achieved by fine-tuning the fully connected layers using the ImageNet weights initialization. To boost the performance of the network we conducted two additional experiments. In the first one we used data augmentation to increase the training entries and in the second one we proposed a mixed architecture (convolutional layers from ResNet-50 and fully connected layers from VGG-16, including dropout with rate 0.5 between them). The following data augmentation transformations were used:

- Rotation range of 20º
- Width shift range of 0.25
- Height shift range of 0.25
- Shear range of 0.25

We discarded the use of random horizontal and vertical flip because these transformations do not necessarily produce new valid images for all the classes of traffic signs.

In all our experiments on object recognition we used RMSProp with learning rate 10^{-4} and 30 epochs for training to optimize the weights of the networks.

IV. Object Detection

In this module our objective was to detect objects of interest involved in driving scenarios, such as traffic signs, pedestrians or vehicles. Given a series of input images, we tested different models that are capable of localizing the items of interest and assigning them a class with a certain confidence score. Examples of object detection ouput are shown in Fig. 4.

Our first experiments consisted in testing an already implemented version of the YOLO architecture. In particular, we used the YOLOv2 variant, which proposes various improvements to the original method [21]. We first used the TT100K detection dataset for training with the objective to detect traffic signs (45 classes considered); and later we used the Udacity dataset to detect pedestrians, cars and trucks (3 classes). In both cases the weights pre-trained on ImageNet were used to initialize the model, and then all layers were re-trained using the RMSProp optimizer with learning rate 10^{-4}. 10 epochs were used for TT100K detection and 40 for the Udacity.

After our experiments with YOLOv2, we trained a Faster R-CNN architecture. To this end, we used the open-source Keras implementation of the model available in: `https://github.com/yhenon/keras-frcnn`. Again, we used the weights pre-trained on ImageNet for initialization, and then re-trained all the layers for 30 epochs using the Adam optimizer with learning rate 10^{-5}. The optimization of Faster R-CNN follows an iterative scheme that alternates fine-tuning for region proposal, performed by the Region Proposal Networks (RPNs), and fine-tuning for object detection, performed by a state of the art CNN. Regarding region proposal, the regression loss defined in [16] is used for the learning of the bounding box coordinates and the binary cross-entropy is used for the learning of object/non-object probabilities. The loss employed for classification is the categorical cross-entropy.

Note that the Faster R-CNN implementation that was used allowed to choose between VGG-16 and ResNet-50 to cover the classification task. In the case of the ResNet-50, 40 conv. layers are shared between RPNs and the classifier, while for VGG-16 the number of shared conv. layers is 13. Given the superior performance of ResNet-50 with respect to VGG-16, validated in section III, we employed ResNet-50.

Fig. 4: Examples of object detection output for TT100K (first row) and Udacity (second row). The detected items are assigned a label and a class probability.

V. Semantic Segmentation

Semantic segmentation is a dense prediction problem, where each pixel is assigned a class label, instead of assigning a single label to the whole image. This is a task of extreme importance in autonomous driving, since vehicles must be able to delimit objects for a proper understanding of the scene (e.g. to keep track of where the road ends and the sidewalk begins).

We started by training and testing a FCN8 architecture. In this model, the final pixel-wise prediction is done after the fusion of the feature maps output associated to the VGG blocks 3-4-5, so resolutions up to a factor of 8 with respect to the original image are considered ($2^3 = 8$, hence the name of the FCN8). ImageNet weights were used to initialize the VGG-16 blocks of FCN8 and then we re-trained all layers. A limit of 1000 epochs was set, with early stopping callback on the Jaccard coefficient (i.e. IoU). Approx. 100 epochs were enough to converge. RMSProp optimizer with learning rate 10^{-4} and batch size equal to 5 were used for weight optimization. Additionally, a learning rate scheduler was activated to decrease the learning rate at each batch following the polynomial model from [19]. With this setting we trained FCN8 using CamVid and the KITTI segmentation dataset (12 classes in both cases: *sky*, *building*, *column*, *road*, *sidewalk*, *tree*, *sign*, *fence*, *car*, *pedestrian*, *cyclist*, *void*).

We also implemented two versions of SegNet, the *SegNet-VGG16* and *SegNet-Basic*. The SegNet-VGG16 is the complete model, with 13 encoders (corresponding to the VGG-16 conv. layers) and 13 decoders. SegNet-Basic is a reduced and compact variant proposed in [10], with just 4 encoders and 4 decoders. We used the same training strategy described above, with the difference that in this case we trained both nets from scratch (without ImageNet weights initialization). The results with CamVid and KITTI are shown in Table III.

To train SegNet-VGG16 with KITTI we used Adam optimizer and increased the learning rate to 0.01. This decision was motivated by the fact that the authors of the net used a larger learning rate (0.1) than our initial value (10^{-4}). In this experiment we also used a callback to decrease the learning rate at specific epochs (20, 40, 60) by a decay rate of 2.

Finally, we also implemented the U-Net architecture described in [11], that has a total of 23 convolution layers (4 of which are transposed, as part of the decontracting path). Following the authors advise, we used data augmentation (rotations, width-height shift and shear transformations) at training time, with Adam optimizer and learning rate of 10^{-4}.

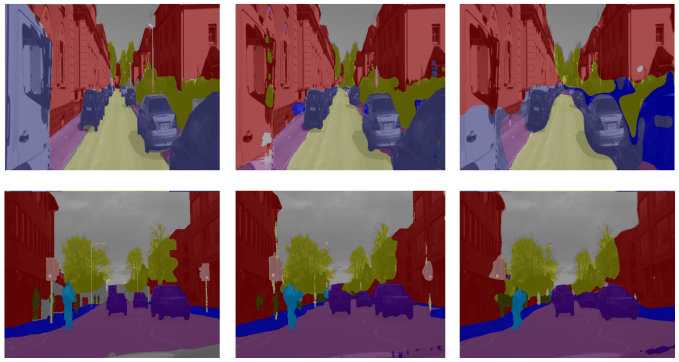


Fig. 5: Examples of segmentation results from the KITTI benchmark (first row) and CamVid (second row). Ground truth is shown in the first column. FCN8 and SegNet-VGG16 predictions are shown in the second and third columns respectively

VI. Results

A. Object recognition

Image resize of 224x224 with mean subtraction and standard deviation normalization for preprocessing stood out as the most suitable method to feed the proposed architectures (95.9% accuracy on test set with VGG-16). The alternative approach, image resize of 256x256 followed by a random crop of 224x224 performed slightly worse (94.2%). The learning transfer from the TT100K to the BelgiumTS dataset with VGG-16 was remarkably successful (93.8% of accuracy), implying that the deep learned features from TT100K generalize nicely for traffic signs of other countries.

Excellent results were also obtained for the KITTI dataset, which shows that our training process from scratch was valid for different datasets. In the case of the KITTI dataset we did not go for fine-tuning using the weights pretrained with TT100K, since the classes from KITTI are not traffic signs anymore. Instead, we fine-tuned the VGG-16 using the ImageNet weights. As it is shown in Table I, slightly better accuracy with respect to train from scratch was achieved.

In any case, the best results for the TT100K dataset were obtained using the ResNet-50, which is no surprise considering that the superior performance of this net with respect to the VGG-16 was already demonstrated in the ILSVRC. The initial result with the ResNet-50 was already impressive (97.3% of accuracy), but data augmentation was still useful to boost performance (97.7%). Fine-tuning using the ImageNet weights also improved accuracy (98.2%). Finally, our experiment with a mixed architecture (convolutional layers from ResNet-50 and fully connected layers from VGG) obtained a 96.9% of accuracy. Even if it did not perform better than ResNet-50, it can be seen as an improvement with respect to VGG-16.

Network	TT100K	BelgiumTS	KITTI
VGG-16	95.9%	-	96.4%
Fine-tuned VGG-16	-	93.8%*	97.1%
ResNet-50	97.3%	-	-
Fine-tuned ResNet-50	98.2%	-	-
ResNet-50 (data aug.)	97.8%	-	-
Mixed architecture	96.9%	-	-

TABLE I: Object recognition results. Accuracy for the TT100K and BelgiumTS datasets evaluated on the test sets. Accuracy for the KITTI dataset evaluated on the validation set, since the test set is not public. *Instead of using ImageNet weights as initialization for the fine-tunning, the weights pretrained on TT100K were used.

B. Object detection

The results obtained for object detection are presented in Table II. As expected, the frames per second (fps) confirm that YOLO (∼68fps) is much faster than Faster R-CNN (∼3fps). In general terms, the F-scores obtained for the TT100K detection dataset reached 40-50%, while remarkably lower values were obtained for the Udacity dataset. A reasonable explanation to this fact is that the train and test splits of the Udacity dataset consist of two driving sequences taken in singularly different locations and lighting conditions (see Fig, 6). It is consequently difficult that the models generalize properly in this case, which results in a disappointing performance on the test set. To overcome the difference in appearance between both sets we tried using data augmentation at training time with YOLO (using the same transformations described in section III), which increased recall but affected negatively the precision, resulting only in a slight increase of the F-score.

Faster R-CNN provided similar results to YOLO (better in the case of Udacity), but we suspect that it could do much better if we increased the number of epochs for training, since the authors of the implementation that we re-adapted suggested a default value of 2000 (with callbacks) and we only used 30 epochs. In this sense, only 10 epochs were used to train YOLO on TT100K, so maybe these results could be improved with a larger amount of epochs as well. Parameter optimization would most probably be also helpful to boost the performance of the models. For Faster R-CNN we adjusted the detection threshold[1] to 0.5 to keep compromise between precision and recall (the default value used for the YOLO model was 0.3), but more parameters should be ideally tuned (optimizer, learning rate, threshold for non-maximum suppression, etc.). We did not conduct more experiments involving a larger amount of epochs or parameter optimization due to time constraints. Also, as an alternative to YOLO, Tiny YOLO (a smaller and more compact variant of the model) could be used to perform much faster detection at the expense of a small decrease in accuracy.

[1]The detection threshold is the minimum class probability for a bounding box to be considered as an object candidate.

Network	TT100K detection / Udacity			
	F-score	Precision	Recall	fps
YOLO	46% / 16%	62% / 17%	36% / 16%	68 / 65
YOLO *	48% / 17%	59% / 15%	40% / 19%	69 / 67
F. R-CNN	40% / 26%	36% / 44%	44% / 18%	3 / 3

TABLE II: Object detection results. F-score, precision, recall and fps evaluated on the test sets of the TT100K and Udacity datasets. The abbreviation F. R-CNN refers to Faster R-CNN. *Data augmentation was used to improve performance.

Fig. 6: Images from the train set (first row) and the test set (second row) of the Udacity dataset.

C. Semantic segmentation

The results for semantic segmentation are summarized in Table III. Notice that global pixel accuracy may not be 100% reliable to assess how well the systems perform, because the classes are not balanced. Most pixels belong to *sky*, *building* or *road* classes, implying that we could misclassify pedestrians or cyclists and still have high accuracy. In this sense, the average Jaccard coefficient or Intersection over Union (IoU) gives a deeper insight on how precise the segmentation really is.

FCN8 was the method that produced the best results for both CamVid and KITTI datasets (highest pixel accuracy and mean Jaccard coefficient), followed by SegNet-VGG16. It makes sense that the two nets have similar results, since their contracting paths are almost the same (the VGG-16 conv. blocks). It also seems reasonable that they performed a little bit worse with KITTI, because only 100 training images were available for this dataset (less than a third part of CamVid).

By taking a closer look to the segmented images we found that SegNet-VGG16 was producing smoother borders than FCN8, that captured sharp contours better (for instance, check the wheels of the cars in Fig. 5). This is not surprising given that our SegNet implementation, for the sake of simplicity, undoes pooling via upsampling with replication. As future work, we intend to incorporate unpooling by reusing the indices from the SegNet encoders with the objective to improve contours.

After all, considering our non-optimal unpooling and the fact that we trained SegNet-VGG16 from scratch (without ImageNet weights initialization), we believe that the results with SegNet are promising. Also, SegNet worked at 19.11 fps at test time, which was faster than FCN8 (20.47 fps). SegNet-Basic was even faster to train and test, but decreased around 5% in the metrics with respect to the full model.

Finally, the performance of U-Net was way below the rest of the models, which suggests that there could be some issue with our code that requires further checking.

Network	TT100K detection / Udacity	
	Pixel accuracy	Mean Jaccard (IoU)
FCN8	86.7% / 83.1%	49.0% / 42.9%
SegNet-VGG16	81.4% / 76.7%	39.3% / 35.3%
SegNet-Basic	76.6% / -	35.4% / -
U-Net	42% / -	8% / -

TABLE III: Semantic segmentation results. Pixel accuracy is the number of correctly classified pixels over the total (for the whole set). Mean Jaccard is the average intersection over union. The networks were evaluated on the test set of the CamVid dataset and the valid. set of the KITTI benchmark. Due to time constraints, in the case of SegNet-Basic and U-Net we only carried experiments with CamVid.

VII. Conclusion

In the context of autonomous driving, we have implemented and tested multiple deep learning models for the tasks of object recognition, object detection and semantic segmentation.

For object recognition we employed popular networks like VGG-16 and ResNet-50, which also play a fundamental role in many other architectures. In this sense, ResNet-50 was used for the classification part in the Faster R-CNN for object detection; and VGG-16 was used in the contracting paths of the FCN8 and SegNet models for semantic segmentation.

ResNet outperformed VGG for object recognition, achieving an excellent 98% of accuracy via fine-tuning on ImageNet weights. Results with object detection have been rather modest, with YOLOv2 achieving a maximum F-score below 50%, while semantic segmentation provided promising outputs, with a Jaccard coef. of 49%, and a 87% pixel accuracy for FCN8.

Several experiments were carried to improve the performance of the different architectures and exploit the wide range of tools and possibilities that deep learning offers. Such experiments involved learning transfer, data augmentation, architecture modifications, and parameter tuning among others.

While it is clear that there is still room for improvement, the present study covered a broad range of concepts and decidedly proved the potential of deep learning for a hot topic of research in Computer Vision as complex as autonomous driving.

References

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

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[3] Z. Zhu, D. Liang, S. Zhang, X. Huang, B. Li, and S. Hu, "Traffic-sign detection and classification in the wild," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[4] R. Timofte, K. Zimmermann, and L. Van Gool, "Multi-view traffic sign detection, recognition, and 3d localisation," *Machine Vision and Applications*, vol. 25, no. 3, pp. 633–647, 2014.

[5] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? The KITTI vision benchmark," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2012, pp. 3354–3361.

[6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[7] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 91–99.

[8] "Udacity dataset," https://github.com/udacity/self-driving-car/tree/master/annotations, 2017, [Online; last accessed 18-March-2018].

[9] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 3431–3440.

[10] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[11] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical Image Computing and Computer-assisted Intervention*. Springer, 2015, pp. 234–241.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[13] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2009, pp. 248–255.

[14] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," *arXiv preprint arXiv:1709.01507*, 2017.

[15] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014, pp. 580–587.

[16] R. Girshick, "Fast r-cnn," *arXiv preprint arXiv:1504.08083*, 2015.

[17] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[18] 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.

[19] L.-C. Chen, G. Papandreou, I. Kokkinos, and K. Murphy, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 4, pp. 834–848, 2018.

[20] Z. Wu, C. Shen, and A. Hengel, "Wider or deeper: Revisiting the resnet model for visual recognition," *arXiv preprint arXiv:1611.10080*, 2016.

[21] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," *arXiv preprint*, 2017.