R-FCN BASED OBJECTION DETECTION FROM PASCAL VOC DATASET

Minaoar Hossain Tanzil, Aadharsh Hariharan

University of Calgary

ABSTRACT

Object detection is a growing field in computer vision where deep learning models are continuously used and improved. This paper demonstrates object detection technique using a prominent deep learning architecture named Region-based Fully Convolutional Neural Network (R-FCN). In order to train and test the neural network, a standardized image dataset - VOC (Visual Object Classes) was used. The final model was used to detect vehicles from Lyft 3D autonomous vehicle detection dateset. The model was trained in Google Colab Pro with GPU and found to have 74.68% mean average precision which remains congruent with the experiment from the original authors.

Index Terms— Object Detection, Deep Learning, Convolutional Neural Network (CNN), Region-based Fully Convolutional Network (R-FCN), Region of Interest (RoI), Region Proposal Network, Convolution, Pooling.

1. INTRODUCTION

Object detection and image classification both requires classification of object inside the image whereas object detection also need to identify the localization of the object. As a result of this similarity, many recent object detection frameworks starting from R-CNN [1] use convolutional neural networks which proves to achieve high accuracy in image classification. But still, because of the localization problem, only CNNs does not provide enough accuracy in object detection. In the paper of R-FCN [2], the authors came up with a novel approach of position sensitive score map after the fully convolutional layer. In this experiment, we use R-FCN for object detection in VOC (Visual Object Classes) data. VOC is standard image dataset by PASCAL project [3].

VOC was first introduced in 2007 and contained around 5,000 training and 5,000 testing images. Subsequently, there was another dataset released in 2012 by VOC which contained around 17,000 training images. As the authors of R-FCN performed their analysis using VOC 2007 and 2012 datasets both, in our experiment we also generate the object detection model using the joint dataset. The model is then applied to Lyft 3D object detection for autonomous vehicles dataset [4].

The replication package for this project is made public in GitHub repository at the location https://github.com/minaoar/ucalgary-enel45.

Thanks to Dr. Henry Leung and Ms. Uzma Ahmed Din for their continuous guidance.

2. RELATED WORKS

2.1. Predecessors of R-FCN

Object detection has received a great attention and progress in last two decades in the computer vision arena [5]. Starting with AlexNet, deep convolutional networks achieved higher accuracy in image classification and thereafter R-CNN [1] was the first prominent object detection technique to use the CNN. Following the success of R-CNN, there came Fast R-CNN [6] and Faster R-CNN [7]. Among all these object detection processes, Faster R-FCN had the state of the art object detection accuracy in VOC dataset (73.2% mAP in 2007 and 70.4% in 2012 dataset).

But as argued by the authors of R-FCN, there is an 'unnatural' RoI pooling layer inserted between two sets of convolution layers of Faster R-CNN. This RoI-wise subnetwork improves accuracy performance at the cost of lower speed, and this dilemma comes from the fact that CNN-based image classification is translation invariant whereas translation variance is an important factor in object detection.To address the issue of translation variance, the R-FCN constructs a set of position-sensitive score maps by using a bank of specialized convolutional layers.

One important similarity among all the above frameworks is that they detect objects in two steps. In the first stage, there is region proposal network (RPN) to generate regions of interest, and in the second stage, RPNs are sent down the pipeline for object classification and bounding box regression.Though such models have higher accuracy, but they are also slower compared to another category of single step object detection [8].

2.2. Single Stage Object Detection Frameworks

On the contrary, there are single stage object detection frameworks presented since introduction of YOLOv1 [9]. In YOLO, the neural network divides the image into regions and predicts bounding boxes and probabilities across all the

Fig. 1. Importance of translation variance in object detection

image classes for all the regions simultaneously. Subsequently,the authors of YOLO came up with their improved versions of YOLOv2, YOLOv3, and YOLOv4 [10] [11] [12].

The other early period single stage framework is SSD [13].SSD introduced multi-reference and multi-resolution detection techniques that significantly improves the detection accuracy for small objects. While YOLOv1 and SSD were faster than two-stage counterpart, they still lagged the detection accuracy until RetinaNet was introduced [14]. They proposed to reshape the standard cross entropy loss such that the loss assigned to well-classified images are reduced. One of the most recent noteworthy single stage object detector is EfficientDet [15].

Similar to their EfficientNet framework for image classification, here they also proposed family of detectors (EfficientDetD0 to EfficientDetD7), a compound scaling method to uniformly scale the resolution, depth, and width for all backbone, feature network, and box/class prediction network simultaneously [16].

3. OVERVIEW OF R-FCN

3.1. Key Idea and Motivation

The major motivation of R-FCN is the importance of translation variance in object detection unlike image classification problem as depicted in Figure 1. If a car's position in the image is transformed or displaced, it's detection outcome may vary.

The key idea of R-FCN is to keep only convolutional layers to create the image feature maps. R-FCN changes the last convolution layer so that it generates a set of position sensitive score maps for each object class as depicted in Figure 2. In this specialized layer, there are distinct K^2 convolutional filters for every classes of the dataset. Every filter is used for a specific region (top-left, bottom-right etc.) of the input from the previous layer.

3.2. Backbone Architecture

The proposed R-FCN is based on ResNet-101 [17] whereas other CNN architectures such as VGG was also possible [18]. ResNet-101 has 100 convolutional layers followed by global average pooling and a 1000-class (for ImageNet classification of 1000 classes [19]) fully connected layer (*fc* layer). In the R-FCN, the average pooling layer and the *fc* layers are removed; only the 100 convolutional layers are kept which computes the feature maps. After these 100 convolutions, two more layers are added - in the first layer, there are 1024 filters with kernel size (1×1) to reduce the dimensions and it is followed by $K^2(C+1)$ numbers of filters where each kernel is $[1 \times 1]$. Author of R-FCN often ignored the dimension reduction layer in their diagrams and layer calculations, and this paper also reflects the same. The last layer with $K^2(C+1)$ filters are used to generate the position-sensitive score banks and hence would be referred as scoring layer. In parallel to the R-FCN layers, there is a block of RPN (Region Proposal Network) layers after the 100 layers of ResNet. These RPN layers have kernel $[1 \times 1]$ and shares the full feature map with the R-FCN network. All these layers are shown in details in Figure 2.

3.3. RoI Scoring by Position-specific Scoring Banks

The most crucial and unique proposition of R-FCN is the position-based voting of proposed regions (RoI). As the last convolution layer of R-FCN generates the K^2 scoring maps for each class, these scoring maps are mostly activated in the regions where the actual object of each class resides.

Each RoI candidate is divided into $(K \times K)$ grids. Each grid (i,j) is overlapped with exactly the (i,j)-th score map from each class. Then the score map values overlapping with the (i,j)-th grid is averaged to generate the pooling value as given in Equation 1. In this manner, every grid's average pooling value is calculated and there are total K^2 average pooling values for each class, C which is averaged for generating the final voting score for class C as depicted in Equation 2. All voting scores of all classes are passed to softmax equation (Equation 3) to generate the probability of the classes.

$$r_c(i,j|\Theta) = \sum_{(x,y)\epsilon bin(i,j)} z_{i,j,c}(x + x_0 + y + y_0|\Theta)/n \quad (1)$$

$$r_c(\Theta) = \sum_{i,j} r_,c(i,j|\Theta) \quad (2)$$

$$s_c(\Theta) = e^{r_c(\Theta)} / \sum_{(c)'=0}^{C} e^{r_{c'}(\Theta)} \quad (3)$$

The pooling process is depicted in the Figure 2. There are two RoIs - one RoI (upper one in the figure) overlaps well with the target object and when the corresponding overlapping pixel values from mapped score bank is chosen, there is fair amount of activation as can be seen from the bright boxes (in image, higher value is represented by near-white or brighter color). Here each box inside the 3×3 grid is calculated using Equation 1. But the bottom RoI is shifted right to the person object and as a result when the overlapping values

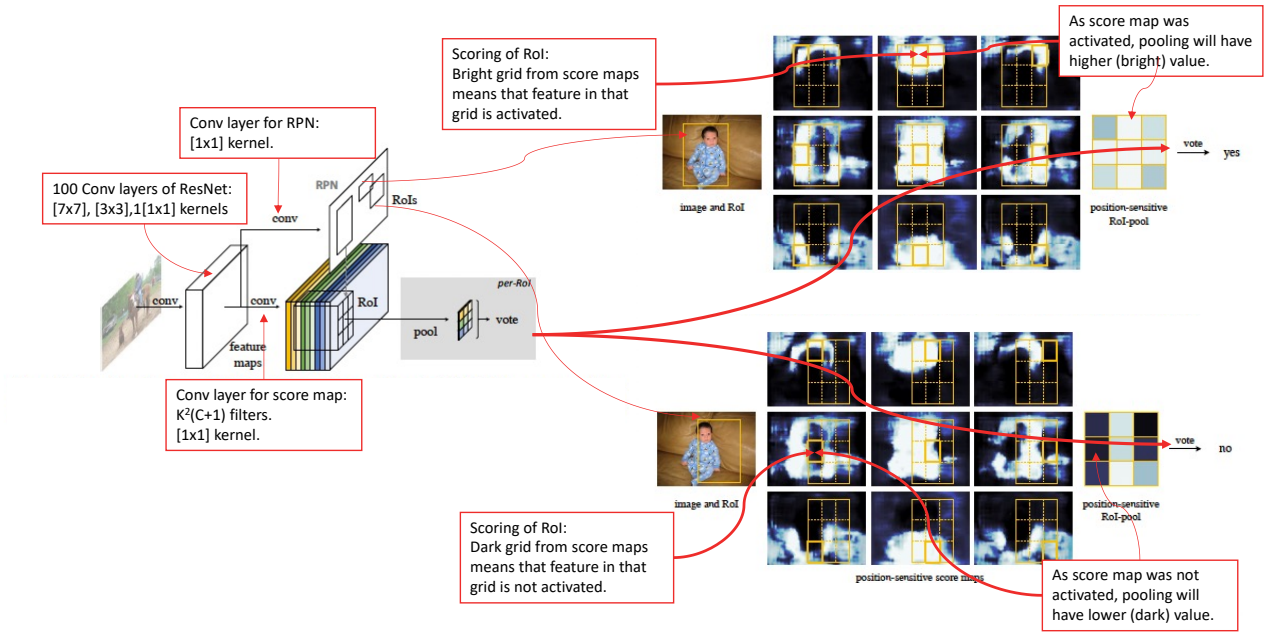


Fig. 2. Architecture of R-FCN along with an RoI pooling example

from the score bank is taken, there is less activation, represented by darker grid values. So when this 3×3 grid will be averaged according to Equation 2, it will have lower value compared with the upper pooling grid and will be discarded as a candidate RoI.

4. MATERIALS AND METHODS

In this experiment, two sets of data is used, the training and testing data is taken from VOC 2007 and 2012 versions. Finally, the inference is applied on Lyft 3D object detection dataset.

4.1. Dataset

4.1.1. Visual Object Class (VOC) Dataset

In VOC dataset, there are four parent categories and 20 classes under those categories as given in Table 1. VOC 2007 has 5011 training images, 4092 testing images, and VOC 2012 has 17,124 images. In our experiment, for training and validation more than 26K images have been used by combining VOC 2007 and VOC 2012 data. The testing data from 2007 dataset has been used as validation data in the experiment. Sample images from VOC dataset is shown in Figure 3

Vehicle	Animal	Indoor	Person
Buses	Cats	Bottles	Person
Cars	Dogs	Chairs	
Bicycles	Cows	Dining-tables	
Motorbikes	Horses	Sofas	
Trains	Sheep	TV/Monitors	
Boats	Birds	Plotted Plants	
Aeroplanes			

Table 1. VOC Object Classes

(a) Cars (b) Motorbike, Person

(c) Horse, Person (d) Sofa

Fig. 3. Samples from VOC Dataset

```
ID: 1841b7895e7163a2c1be87e8c1740e759a910cd59157b8a
c8db8c83aac0a1ae3
Annotations:
848.9346793681768 2587.2870557176507 -20.1599565558
6927 1.963 4.345 1.749 -1.887128017837603 car
```

(a) Annotations of Lyft dataset is identified by unique IDs. Under every ID there are multiple annotations of object from multiple images.

```
ID: db8b47bd4ebdf3b3fb21598bb41bd8853d12f8d2ef25ce76ed
d4af4d04e49341
1 images/host-a101_cam0_1242144892450000006.jpeg
5 images/host-a101_cam2_1242144892483330006.jpeg
6 lidar/host-a101_lidar2_1242144892502687076.bin
```

(b) The training data contains Lidar data files and image files together.

(c) The database and relationship between annotation IDs and multiple image or Lidar files are put in JSON format.

Fig. 4. Lyft data format

4.1.2. Lyft 3D Object Detection Dataset

Lyft has an object detection competition in Kaggle for their autonomous vehicles. This dataset contains more than 150K training images and around 190K test images. It not only contains images in jpeg format, but also contains LIDAR data. They have their own custom software development kit to work with Lyft dataset. Since this dataset is not used in any academic literature mentioned in related works section, preprocessing this dataset for training a model like R-FCN was a non-trivial job. Here is a glimpse of the Lyft data format in Figure 4.

4.2. Methodology

4.2.1. Primary Training Method

In order to train and test the VOC dataset, a public repository developed by researchers at East China University of Science and Technology was used [20]. The source code is developed upon TensorFlow using Python3 libraries, and runs on GPU. As a result, it was possible to run the total training and testing in Google Colab. Since it takes hours to train thousands of images, Colab Pro version was necessary to prepare the models. The process flow is depicted in Figure 5.

The training was done for 110K iterations. In every batch, one image was provided, since it was running in GPU, effective batch size was 8. The minimum threshold to qualify the proposal RoI was 0.5 IoU (intersection over union) compared with the ground truth object. To remove multiple bounding box of a single object, non-max-suppression (NMS) technique was used with a threshold value of 0.3. For classification of object classes, minimum confidence level 0.7 was considered. All kernels added for R-FCN and RPN networks had size of 1×1. To divide every RoI in 3×3 grids of positions, there were 9 score maps per image class. So the value K was set to 3. As a result, for the VOC dataset which had 20 classes and 1 background, there was total 3x3x(20+1)=189 filters created in the R-FCN final convolutional layer.

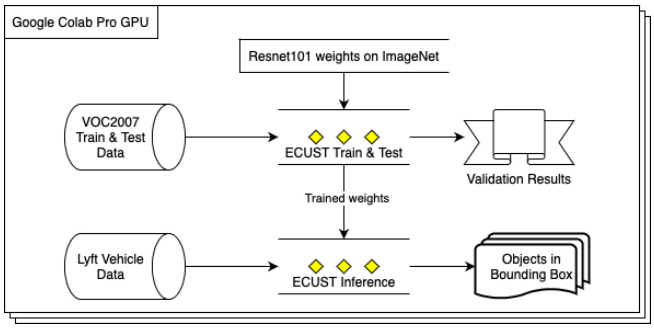


Fig. 5. Process flow for the model training and inference using Google Colab Pro

4.2.2. Additional Training

In addition to the GPU supported TensorFlow implementation, another CPU-only TensorFlow implementation from researchers of University of Chinese Academy of Science was experimented [21]. In this training, 1K images from VOC 2007 dataset was used. All hyper-parameters (kernel size, thresholds etc.) were same as the primary experiment. As this implementation did not utilize GPU, it was run on local machine instead of any cloud platform as shown in Figure 7. In subsequent sections, unless otherwise mentioned explicitly, all results are reported from the primary experiment with GPU.

4.2.3. Additional Contribution

Though we found only one GPU and Python3 library compatible TensorFlow implementation from researchers of East China University of Science and Technology, there were some limitations in their implementation. First of all, their implementation of NMS (non-maximum suppression) used native C code and it had poor performance as depicted in Figure. As can be seen that there were still dozens of candidate bounding boxes for a single object. We improved this NMS by using latest TensorFlow library. To make this change, we had to debug their implementation in very detail and change their core models.

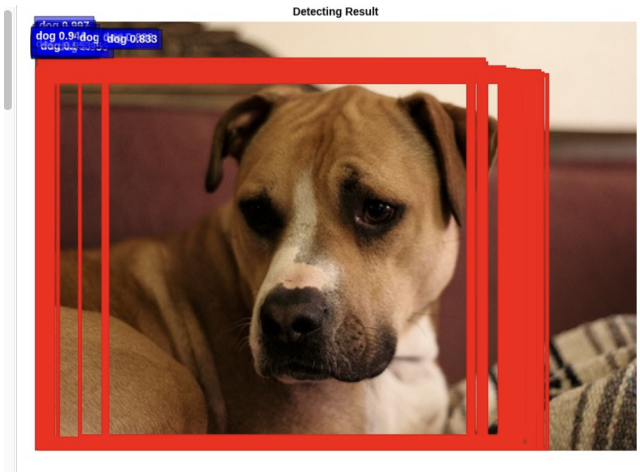


Fig. 6. Previous insufficient NMS (non-maximum suppression) implementation that was improved by us. In all other images in this paper, we used improved NMS implementation and that is why there is no such multi-boxes per image.

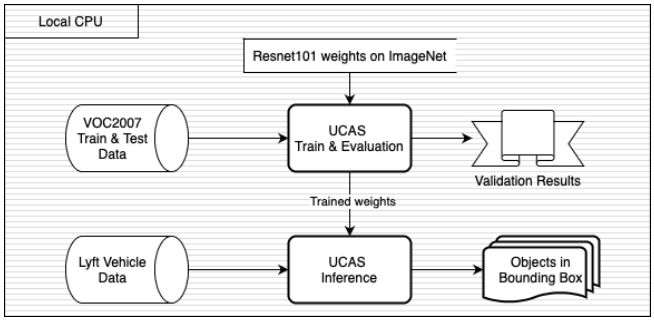


Fig. 7. Process flow for the additional experiment using CPU

Secondly, in all cases of training in the original paper, the authors of R-FCN always trained with both VOC 2007 and VOC 2012 dataset. But in the chosen implementation only VOC 2007 was supported which provided mAP of around 72.6% only in our initial experiment result. So we had to analyze their system to support VOC2012 dataset, because the configuration with multiple and large dataset is different according to the original implementation by the R-FCN authors. After adding VOC2012 images to training dataset, as we will be seeing in the results section, the mAP increased by around 2% to 74.67%. In all of our results, we presented only the enhanced training dataset.

5. RESULTS AND DISCUSSION

5.1. Results

The GPU-based training took nearly 6.5 hours to train on around 27K image set whereas the additional experiment with CPU took 20 hours just to train on 1K images. The results are

Fig. 8. Detected vehicles

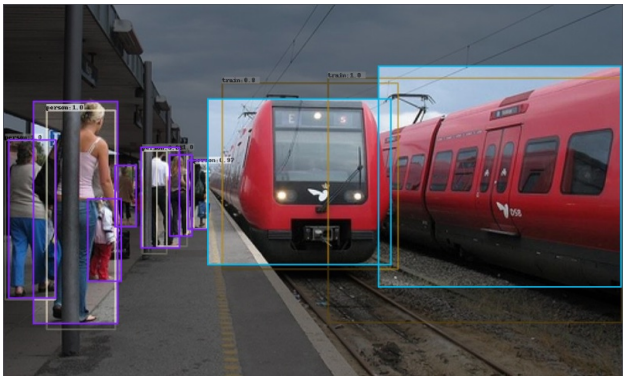


Fig. 9. Bounding box of ground truth and detected objects

summarized in Table 2. The class-wise average precision is listed in Table 3.

The detection results from the experiment are displayed in Figure 8 and the intersection over union among the detected objects and ground truth bounding boxes are sampled in Figure 9.

5.2. Discussion

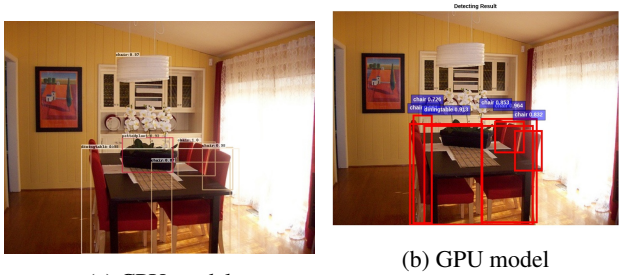
The mean average precision found in the primary experiment with GPU is congruent with the original results of the R-FCN paper. In the original paper, the detection accuracy with training data of VOC2007 and VOC2012 was 77.6% with multi-scaling (which means a single image was scaled to multiple resolutions such as 600, 800 etc.). But in our experiment, the

	Primary Experiment (GPU)	Additional Experiment (CPU)
Mean Average Precision	0.7468	0.82
Training Time	6 hours 20 min	20 hours
Training-Validation Size	26,227	1000
Labelling Performance	Does not label wrong objects.	Detects more objects, but labels wrong.

Table 2. Summary of Results

Class Name	Average Precision
aeroplane	0.7817
bicycle	0.7968
bird	0.7692
boat	0.6597
bottle	0.6209
bus	0.8322
car	0.8512
cat	0.8737
chair	0.5830
cow	0.8021
dining-table	0.6156
dog	0.8499
horse	0.8046
motorbike	0.7768
person	0.7814
potted-plant	0.4924
sheep	0.7470
sofa	0.7244
train	0.8049
tv/monitor	0.7679

Table 3. Class specific average precision



(a) CPU model

(b) GPU model

Fig. 10. Comparison of classification between the fully trained GPU-based model and limited CPU-trained model. (a) CPU model labeled the lamp as chair, whereas (b) GPU model ignored it.

same dataset was used for training and validation, but multi-scaling was not used, only single scaling was used. As a result, the mAP of our experiment is 74.68%. It means that using multi-scaling, mAP can be increased up to 1.9%. This is an important identification of the experiment.

The next important observation is that although we attempted to train the model with a CPU-only version and trained with only 1000 images, it gives some wrong classification. For example, in the Figure **??**, the limited trained CPU model detects the overhead lamp as a chair. Whereas as shown in the Figure 10b, the fully trained GPU model does not perform any classification of the overhead lamp and thus does not give wrong labelling. It is noteworthy that in the training classes, there was no class for lamps, so not detecting a lamp is acceptable for the models.

Finally, the lack of latest implementation of R-FCN network in major libraries such as TensorFlow, PyTorch or Keras is noteworthy. Though there are implementations by the original authors, those were updated 2/3 years ago around 2016 2018 [22] [23] [24]. As a result, they are sometimes not compatible of latest Python3 libraries. There is only implementation by a major library 'Detectron' by Facebook (now Meta) [25]. But Detectron also uses incompatible libraries with Python3. In fact, we found only one TensorFlow implementation of the R-FCN algorithm. But that implementation also did not support training with VOC 2012 data. We had added support for training with multiple dataset of VOC2007 and VOC2012 to that implementation.

One reason for not having enough latest implementation of R-FCN can be attributed to two factors. First of all, accuracy-wise it was close to then state-of-the-art algorithm of Faster R-CNN; it did not promise significant improvement. Its advantage was mainly the faster detection time. Interestingly, the single stage detection algorithms such as SSD and YOLO were also proposed around the same time of R-FCN and those single stage papers were far more promising in quick (nearly real-time) detection performance. As a combination of these two factors - high accuracy of Faster R-CNN and faster detection performance of YOLO or SSD, the usage and implementation of R-FCN might not get further traction in later future. Still, its contribution to location specific score maps and pooling was noteworthy and novel.

6. CONCLUSION

R-FCN proposed significant performance improvement in terms of detection time which was at times up to 20 times faster than Faster R-CNN. In our experiment, we also found similar accuracy and speed for the VOC dataset. Moreover, we were able to improve the GPU-compatible source code to provide additional support of training with VOC2012 dataset. We also fixed the NMS multi-box problem of the implementation. From these observations, it seems that there can be further improvement of R-FCN implementation using latest Python libraries in PyTorch or TensorFlow. Also, it would be noteworthy to compare the performance of latest single stage and two steps algorithms such as YOLO, EfficientDet and the performance of R-FCN.

7. REFERENCES

[1] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[2] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun, "R-fcn: Object detection via region-based fully convolutional networks," in *Advances in neural information processing systems*, 2016, pp. 379–387.

[3] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, June 2010.

[4] Kaggle, *Lyft 3D Object Detection for Autonomous Vehicles*, `https://www.kaggle.com/c/3d-object-detection-for-autonomous-vehicles/data`, 2019, Last accessed on 16 December 2021.

[5] Zhengxia Zou, Zhenwei Shi, Yuhong Guo, and Jieping Ye, "Object detection in 20 years: A survey," *arXiv preprint arXiv:1905.05055*, 2019.

[6] Ross Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[7] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, pp. 91–99, 2015.

[8] Petru Soviany and Radu Tudor Ionescu, "Optimizing the trade-off between single-stage and two-stage deep object detectors using image difficulty prediction," in *2018 20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. IEEE, 2018, pp. 209–214.

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

[10] Joseph Redmon and Ali Farhadi, "Yolo9000: better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7263–7271.

[11] Joseph Redmon and Ali Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[12] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[13] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[14] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[15] Mingxing Tan, Ruoming Pang, and Quoc V Le, "Efficientdet: Scalable and efficient object detection," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 10781–10790.

[16] Mingxing Tan and Quoc Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International Conference on Machine Learning*. PMLR, 2019, pp. 6105–6114.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

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

[19] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al.,

"Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[20] East China University of Science and Technology, *Repository of East China University of Science and Technology*, https://github.com/engineer1109/TensorFlow-R-FCN-for-primary-students, 2019, Last accessed on 16 December 2021.

[21] University of Chinese Academy of Science, *Repository of University of Chinese Academy of Science*, https://github.com/DetectionTeamUCAS/R-FCN_Tensorflow, 2019, Last accessed on 16 December 2021.

[22] Authors of R-FCN, *Official R-FCN Implementation*, https://github.com/daijifeng001/r-fcn, 2016, Last accessed on 16 December 2021.

[23] Yuwen Xiong, *Python Implementation of R-FCN*, https://github.com/YuwenXiong/py-R-FCN, 2017, Last accessed on 16 December 2021.

[24] Computer vision fanatics from Microsoft Research Asia, *Deformable ConvNets*, https://github.com/daijifeng001/r-fcn, 2018, Last accessed on 16 December 2021.

[25] Facebook Research, *Detectron*, https://github.com/facebookresearch/Detectron/blob/main/detectron/modeling/rfcn_heads.py, 2017, Last accessed on 16 December 2021.