SINGLE-SHOT DETECTOR WITH MULTIPLE INFERENCE PATHS

Shoufa Chen, Xinggang Wang

Institute of AI, School of EIC, Huazhong University of Science and Technology

ABSTRACT

In this paper, we investigate the problem of resource-constrained object detection using deep learning, which is a challenging problem in real-world applications. To address this problem, we propose a single-shot detector with multiple inference paths based on a multi-scale DenseNet. Experiments are carried out on the PASCAL VOC and COCO datasets, and results show that, with significant computation reduction, our detection network obtains comparable performance corresponding to the single-shot detectors, such as YOLO and SSD.

***Index Terms*—** Object detection, resource-constrained, deep networks.

1. INTRODUCTION

In recent years, deep convolutional neural networks (CNN) have witnessed great successes in computer vision tasks, such as image classification, object detection, semantic segmentation, image generation, and visual tracking. Among these, object detection plays an important role in various industrial applications such as autonomous driving, surveillance, so it has attracted much research attention.

Current object detection methods, such as Faster-RCNN [1], SSD [2], YOLOv3 [3] and anchor-free detectors [4, 5], tend to rely mostly on deep architectures to obtain better accuracies and are so resource-hungry that have greedy computational resources demands as CPU, GPU, electric power, restraining the deployment of object detection applications in more widely accessible devices, such as mobile phones, drones, robots and autonomous driving cars. Therefore, in resource-constrained situations, we should consider more about the practicability of detectors.

Besides, although complex algorithms and deeper architectures are necessary for detecting hard examples, it is enough for the easy examples to just utilize much smaller models whilst obtain decent performance. Cars in Fig. 1 illustrate this scenario. The left image depicts an "easy" example in a distinguishable viewpoint with almost no interference so it is nothing to classify and localize this car for a normal detection model. However, the right image shows "hard" examples as it includes many small cars in various viewpoints. The semantic information enriched by the stack of convolutional networks can benefit the "hard" example detection but it is just a waste of computational and time resource for the "easy" examples which can be detected properly by utilizing some light-weight models, especially at resource-constrained situations such as robotics, mobile phones etc.

Fig. 1. An "easy" image (left) can be properly detected with less computation whereas a "hard" image (right) requires deeper model and expensive computation to detect.

Deep learning in resource-constrained devices has attracted lots of attention recently. The most straight-forward solution is to design light-weight deep networks, such as MobileNets [6], IGCVNets [7], ShuffleNets [8] etc. These methods design specific network architectures which can produce high object recognition accuracies with low computation cost by manually designed rules or semi-automatically learning strategies.

However, the light-weight networks use a fixed inference network all images. In this paper we propose a single-shot detector with multiple inference paths, named as SSD-MIP. Supposing the detection hardness of input images can be determined, easier images can be inferred using the paths with fewer computation and harder images can be inferred using the paths with more computation. SSD-MIP is built on MSDNet [9], which is proposed for image classification with computation resource limits. We integrates the rich feature hierarchies with multi-path single-shot detectors. We carry out experiments on both the PASCAL VOC dataset and the COCO dataset, and report both detection accuracies and computation costs, in comparison with the top performing detectors.

In summary, we propose an efficient object detection network for resource-constrained object detection with the following contributions: (1) We propose a novel object detector which contains multiple inference paths which naturally takes advantage of the multi-scale feature maps in DenseNet for efficient object detection. (2) The proposed network obtains comparable performance corresponding to the state-of-the-art single-shot detectors, such as YOLO and SSD, with significant computation reduction.

2. RELATED WORK

Deep learning-based object detection Based on convolutional neural networks, recent approaches achieve promising accuracies especially by means of region proposal methods which have become prevalent since the dramatic improvement bought by R-CNNs [10, 1]. Single-stage methods skip the proposal step for faster speed, which are utilized to investigate resource limit object detection. SSD [2] assigns anchors of different scales to multiple layers owning different feature map scales throughout the network to explicitly tell anchors to pay attention to predict objects of a certain scale. Different from SSD, YOLO utilizes a single feed-forward network to learn class probabilities and locations directly from the full images. YOLOv2 [11] further improves YOLO [12] in both speed and accuracy aspects by introducing more tricks, such as batch normalization, anchor boxes, fine-grained features, etc.

Resource constrained deep learning Running deep learning models on edge devices, such as smartphones, drones, robots, self-driving cars, surveillance cameras etc, has large demands in real applications. However, deep networks are known for computational expensive and contain a huge amount of parameters. To tackle this problem, there are a large number of research works solve the problem in the following aspects. (1) Network compression and pruning: these methods solve the problem by quantizing the float parameters into fix-point parameters and iteratively delete the unimportant neurons, such as [13, 14]. (2) Light-weight networks, these methods either use sparsely connected convolution operations or search some specific neural architectures that are more more efficient, such as [6], IGCVNets [7], ShuffleNets [8], CondenseNet [15]. The proposed method attempts to solve the resource constrained object detection problem from another perspective by detecting different images using different paths.

3. METHOD

In this section, we introduce our SSD-MIP network in detail. First, we review the recently proposed MSDNet [9], which serves as our feature extraction backbone network. Second, we present the multiple inference path detection network based on the backbone network.

3.1. Backbone Network: MSDNet

As illustrated in Table 1, MSDNet was designed to generate both fine and coarse level features throughout the network depth so early classifiers can be inserted along intermediate layers. And the introduction of the inter-connected layers resorting to dense connectivity suppresses interference among different classifiers.

Input images are first fed to a 7×7 convolution and a 3×3 max pooling layer (both with stride 2), get the transformed features, x_0^0, with both width and height divided by a factor 4 respect to the original image. Then, x_0^0 will enter the first layer of the MSDNet.

The subsequent layers are designed following the main idea of DenseNet [16]. Formally, the output feature map x_ℓ^s at layer $\ell \geq 0$ and scale s is produced by concatenating the transformed features from all of the previous layers at scale s and $s-1$ if possible (i.e., when $s > 1$). The resulting output x_ℓ^s becomes:

$$x_\ell^s = \begin{cases} h_\ell\left(\left[x_0^s, ..., x_{l-1}^s\right]\right) & \text{if } s = 1 \\ \left[\begin{array}{l} h_\ell\left(\left[x_0^s, ..., x_{l-1}^s\right]\right) \\ \\ \tilde{h}_\ell\left(\left[x_0^{s-1}, ..., x_{l-1}^{s-1}\right]\right) \end{array}\right] & \text{if } s \geq 1 \end{cases} \quad (1)$$

where $[\dots]$ denotes the concatenation operator and the transformation are designed following the DenseNet feed-forward fashion, i.e., **Conv**(1×1)**-BN-ReLU-Conv**(3×3)**-BN-ReLU**, producing 16, 32, 64, 64 features for 4 scales respectively.

3.2. Object detection via multiple inference paths

Using the feature extractor MSDNet, we possess multi-scale (along vertical dimension) and various semantic enriched (along horizontal dimension) feature maps, as shown in Table 1, which provide adequate selections for the detector taking as input. In order to further utilize the features generated by MSDNet, different to the original classification task which only uses the coarsest scale at each block, we use 3 scales for each of the object detectors, following the style similar to FPN [17] and YOLOv3 [3] which use a top-down architecture with lateral connections. Formally, we denote the picked features for one detector as $x_\alpha^2,\ x_\beta^3,\ x_\gamma^4$.

In the following of this subsection, we will take one detection path ($\alpha = 10, \beta = 15, \gamma = 20$) for instance to introduce our proposed detection methods and it is easy to generalize to the multiple paths.

For the feature x_γ^4, we append three convolution layers to it, with kernel $1\times1, 3\times3, 1\times1$ and filters $\Gamma^4,\ 2\times\Gamma^4,\ \Gamma^4$ respectively. Each of the convolution layers is followed by a **BN** [18] and **Leaky ReLU** with slope coefficient 0.1 for the negative half-axis. Then we get the candidate feature m_γ^4 which will be sent into the last convolution layer to classify and localize the object in the image. Before that, we will first introduce how to utilize the finer scale feature maps.

After getting the lateral connection result feature $\hat{x}_\beta^3$, with the same scale as x_β^3, we treat $\hat{x}_\beta^3$ totally identically as x_γ^4, i.e. appending the same transformation architecture, except modifying the convolution filters from Γ^4 to Γ^3. This principle is also applicable to the finest scale feature x_α^2.

3.3. Convolutional predictors for object detection

Each feature among $\{m_\alpha^2, m_\beta^3, m_\gamma^4\}$ can produce a fixed set of detection predictions. For a feature layer of size $S \times S$, it can be sent into an essential convolutional layer that is optimized to predict class probabilities and bounding box coordinates. Operating in a sliding-window fashion, the convolutional filters move through each spatial position through

Table 1. Feature channels statistics. The features of **Scale 1** are in the largest scale, and both feature width and height are divided by a factor of 2 for the **Scale 2** corresponding to **Scale 1**, and so on. All horizontal features have the same feature scale. $\Downarrow$ indicates the transition layer. ✗ indicates the features are not necessary to compute and do not exist.

	Stage 1						Stage 2						Stage 3						Stage4					
channel \ layer / scale	0	1	2	3	4	5	6	$6^{\Downarrow}$	7	8	9	10	11	$11^{\Downarrow}$	12	13	14	15	16	$16^{\Downarrow}$	17	18	19	20
1	32	48	64	80	96	112	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
2	64	96	128	160	192	224	256	128	160	192	224	256	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
3	128	192	256	320	384	448	512	256	320	384	448	512	576	288	352	416	480	544	✗	✗	✗	✗	✗	✗
4	128	192	256	320	384	448	512	256	320	384	448	512	576	288	352	416	480	544	608	304	368	432	496	560

the feature map. For each box out of B at a given position, it computes $(4 + 1 + C) \times B$ values to represent the bounding box predictions. Here, B is the number of the anchor box assigned to each position, C representing the classes probabilities the object belongs to, 1 confidence score reflecting the probability that the box contained an object and 4 coordinates offsets encoding the object location. Above all, for a feature map with the size $S \times S$, the output will be a $S \times S \times [(4 + 1 + C) \times B]$ tensor. We use three scale features for a detection path, so the predictions will be $(S_2 \times S_2 + S_3 \times S_3 + S_4 \times S_4) \times [(4 + 1 + C) \times B]$.

3.4. Training objective

For each ground truth box, we first determine which grid the object locates at, then select the best closest anchor at that grid based on the Jaccard overlap. The training objective based on the anchor is defined as follows.

Location loss: The location loss is the coordinate loss, measuring the center and scale of the predict box relative to the ground truth. Formally, it is defined as:

$$\begin{aligned} L_{loc} = &\sum_{i=2}^{4}\sum_{j=0}^{S_i^2-1}\sum_{m\in\{cx,cy\}} \text{BCELoss}(l_{ij}^m,\ \hat{g}_k^m;\ I_{ij}^k) \\ &+\sum_{i=2}^{4}\sum_{j=0}^{S_i^2-1}\sum_{m\in\{w,h\}} \text{L1Loss}(l_{ij}^m - \hat{g}_k^m;\ I_{ij}^k) \end{aligned} \tag{2}$$

where $I_{ij}^k \in \{1, 0\}$ is an indicator denotes whether the default anchor at feature scale S_i and position j is responsible for the k-th ground truth in the image. The location loss only penalizes the matched anchor box. And the **BCELoss** is binary cross-entropy loss defined as: $\text{BCELoss}(l, g; I) = -I \times [g \times \log(l) + (1 - g) \times \log(1 - l)]$

Objectness loss: This loss optimizes the probability of the presentness of the object associated to the anchor. It is defined as follows: $L_{obj} = \sum_{i=2}^{4}\sum_{j=0}^{S_i^2-1} \text{BCELoss}(l_{ij}^{obj},\ \hat{g}_k^{obj};\ I_{ij}^k)$

Classification loss: This loss optimizes the probability of which class the corresponding object belongs to. It is calculated as follows: $L_{class} = \sum_{i=2}^{4}\sum_{j=0}^{S_i^2-1}\sum_{m\in\text{classes}} \text{BCELoss}(l_{ij}^m,\ \hat{g}_k^m;\ I_{ij}^k)$

The training objective for one branch is to minimize the multi-task loss: $L_b = \lambda_{loc} \times L_{loc} + \lambda_{obj} \times L_{obj} + \lambda_{class} \times L_{class}$ where, λ is a trade-off coefficient to balance the three part loss and we set all of them as 1 in our experiments.

Table 2. The feature layers selected for the four paths.

Paths/ Layer	α	β	γ
1	6	6	6
2	10	11	11
3	10	15	16
4	10	15	20

3.5. Inference

For the proposed SSD-MIP network, we pick 4 paths of detectors and the multi-scale features for each path are presented in Table 2. The 4 detection paths are trained at the same time. At inference phase, every single path can complete the detection work independently.

4. EXPERIMENTS

We conduct comprehensive experiments on the widely used PASCAL VOC [19] and COCO [20] datasets, which have 20 object categories and 80 object categories, respectively. We implement the proposed method using PyTorch and our source codes will be released on publication.

The 4 detection paths are based on 12 convolutional feature maps. Following the widely used practices, we use the MSDNet model pretrained on ImageNet classification dataset [21]. The whole detection network is trained with stochastic gradient descent (SGD) solver on over 2 NVIDIA GeForce GTX 1080 Ti GPUs. Unless otherwise specified, we train a total of 32 images per mini-batch (16 images per GPU). All models are trained for 200 epochs with an initial learning rate of 0.01, which is then divided by a factor of 10 at 160 and 180 epochs. The data augmentation strategies mainly follow YOLOv3, including image color distortion, expansion, cropping, and horizontal flip randomly. We use the standard mean average precision (mAP) to measure the object detection performance. We evaluate the number of operations measured by multiply-adds (MAdds), following the standard setting in [6], which is used to measure the computation cost.

4.1. Results on VOC 2007

For PASCAL VOC, we split the dataset following the common strategy that combines the VOC 2007 `trainval` and VOC 2012 `trainval` as the training data (16,551 images) and test on VOC 2007 `test` data (4,952 images). The popular state-of-the-art object detectors are compared, such as Faster RCNN [1], SSD [2], YOLO [11] and R-FCN [22].

Table 3. Comparison of different detectors on PASCAL VOC 2007 `test`. † denotes the result of fine-tuning the COCO detection model on VOC 2007 `trainval` + VOC 2012 `trainval`.

Method	Backbone	MAdds	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
Fast RCNN	VGG16	-	70.0	77.0	78.1	69.3	59.4	38.3	81.6	78.6	86.7	42.8	78.8	68.9	84.7	82.0	76.6	69.9	31.8	70.1	74.8	80.4	70.4
Faster RCNN	VGG16	-	73.2	76.5	79.0	70.9	65.5	52.1	83.1	84.7	86.4	52.0	81.9	65.7	84.8	84.6	77.5	76.7	38.8	73.6	73.9	83.0	72.6
Faster RCNN†	VGG 16	-	78.8	84.3	82.0	77.7	68.9	65.7	88.1	88.4	88.9	63.6	86.3	70.8	85.9	87.6	80.1	82.3	53.6	80.4	75.8	86.6	78.9
R-FCN	ResNet101	-	80.5	79.9	87.2	81.5	72.0	69.8	86.8	88.5	89.8	67.0	88.1	74.5	89.8	90.6	79.9	81.2	53.7	81.8	81.5	85.9	79.9
SSD300	VGG16	31.4B	77.5	79.5	83.9	76.0	69.6	50.5	87.0	85.7	88.1	60.3	81.5	77.0	86.1	87.5	84.0	79.4	52.3	77.9	79.5	87.6	76.8
SSD300†	VGG16	31.4B	79.6	80.9	86.3	79.0	76.2	57.6	87.3	88.2	88.6	60.5	85.4	76.7	87.5	89.2	84.5	81.4	55.0	81.9	81.5	85.9	78.9
YOLOv2 (416)	Darknet-19	17.5B	76.8	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SSD-MIP (path 1)	MSDNet	7.8B	61.3	65.3	72.0	55.4	46.9	32.1	70.9	74.0	74.4	38.9	59.0	60.0	68.1	77.9	71.8	70.2	32.2	58.4	60.3	74.0	64.0
SSD-MIP (path 2)	MSDNet	8.9B	69.7	70.1	78.7	66.6	55.9	38.6	82.2	82.8	86.3	50.4	69.4	66.5	80.0	83.7	76.8	73.7	44.0	68.7	70.4	78.8	70.2
SSD-MIP (path 3)	MSDNet	9.7B	71.4	69.2	79.7	67.4	60.3	41.6	84.1	79.4	87.7	53.9	70.7	70.3	80.8	85.2	78.5	75.6	46.9	68.2	77.5	78.6	72.4
SSD-MIP (path 4)	MSDNet	9.8B	76.4	80.3	80.5	75.5	68.8	61.8	85.1	86.0	87.8	56.2	80.8	74.2	83.5	86.6	84.5	77.4	51.1	77.3	76.3	79.5	75.8
SSD-MIP†(path 1)	MSDNet	7.8B	61.9	65.5	72.3	59.2	48.1	33.5	71.7	74.5	74.4	39.7	62.6	60.5	69.1	77.0	72.3	70.9	34.8	59.5	58.5	72.5	61.1
SSD-MIP†(path 2)	MSDNet	8.9B	70.8	70.5	79.4	66.5	59.5	42.0	84.5	79.6	83.9	51.9	73.4	68.6	80.1	86.0	77.8	76.7	47.7	69.9	71.9	78.2	67.2
SSD-MIP†(path 3)	MSDNet	9.7B	75.1	71.4	80.0	68.9	63.4	56.9	84.9	87.4	87.0	57.3	80.5	74.3	82.2	88.1	79.8	78.5	54.8	73.1	77.9	84.8	69.9
SSD-MIP†(path 4)	MSDNet	9.8B	80.0	87.0	86.1	76.9	72.3	68.1	86.2	87.7	87.2	62.4	86.8	78.3	84.0	88.7	85.6	79.1	57.1	84.8	78.1	86.1	77.2

Table 4. Results on COCO `test-dev`. 'sml', 'mdm' and 'lrg' stand for small, medium and large respectively, and 'mAP', 'AP50' and 'AP75' mean average precision of IOU = 0.5:0.95, IOU=0.5 and IOU=0.75 respectively. `trainval35k` is obtained by removing the 5k `minival` set from `trainval`.

Method	Backbone	MAdds	data	mAP	AP50	AP75	APsml	APmdm	APlrg	AR1	AR10	AR100	ARsml	ARmdm	ARlrg
Faster RCNN	VGG16	-	trainval	21.9	42.7	-	-	-	-	-	-	-	-	-	-
R-FCN	ResNet101	-	trainval	29.9	51.9	-	10.8	32.8	45.0	-	-	-	-	-	-
SSD300	VGG16	35.2B	trainval35k	25.1	43.1	25.8	6.6	25.9	41.4	23.7	35.1	37.2	11.2	40.4	58.4
SSD321	ResNet101	-	trainval35k	28.0	45.4	29.3	6.2	28.3	49.3	25.9	37.8	39.9	11.5	43.3	64.9
YOLOv2	DarkNet-19	17.5B	trainval35k	21.6	44.0	19.2	5.0	22.4	35.5	-	-	-	-	-	-
YOLOv3 (416)	DarkNet-53	32.9B	trainval35k	31.0	-	-	-	-	-	-	-	-	-	-	-
YOLOv3 (608)	DarkNet-53	70.3B	trainval35k	33.0	57.9	34.4	18.3	35.4	41.9	-	-	-	-	-	-
SSD512	VGG16	99.5B	trainval35k	28.8	48.5	30.3	10.9	31.8	43.5	26.1	39.5	42.0	16.5	46.6	60.8
SSD-MIP (path 4)	MSDNet	10.0B	trainval35k	28.9	49.1	30.0	11.8	29.2	41.5	25.2	36.8	38.0	17.4	39.1	54.7

The detectors are evaluated with and without the COCO dataset pretraining. Note that the computation costs (MAdd) for Faster RCNN, R-FCN etc are variable for different sizes of input images; thus their MAdds are not reported; nevertheless, their computation costs are significantly larger than the single-shot detectors, YOLO, SSD, and our proposed SSD-MIP. From the results in Table 3, we have the following observations: (1) The proposed SSD-MIP method based on MSDNet is very computational efficient: it costs less 10 billions of MAdds, which is significantly less than the other detectors. (2) Among the 4 paths of the detectors, deeper detector obtains better performance; the fourth path detector obtains the best performance, 78.3% mAP (without COCO pertraining), which is comparable to YOLOv2, while has a half of computation cost.

4.2. Results on COCO

Finally, we perform experiments COCO to further validate the SSD-MIP performance. COCO contains 80k training examples, 40k validation examples and 20k testing examples (`test-dev`). Following the common settings, we train our SSD-MIP on `trainval35k` set and test on the `test-dev2017` dataset. Because `test-dev2017` and `test-dev2015` contain the same images, the results obtaining from them are comparable. Results are summarized in Table 4. Our SSD-MIP achieves $28.9\%/49.1\%$ on the `test-dev` set, which outperforms the YOLOv2 over a wide range (7.3%) with almost half of the MAdds relative to YOLOv2. And our SSD-MIP is slightly better than SSD512, which requires 10 times MAdds than ours.

4.3. Limitations

Though the upper bound performance is a decent result with regard to both accuracy and computation resource, there are still some limitations requiring further investigation. It is not easy for the SSD-MIP network to determine which detection path is most suitable for the input image. We need to embed a delicate module into our SSD-MIP so that our model has the ability to make a decision on the path number to exit automatically. So how to teach the network to achieve the upbound performance only resorting to raw image information is a challenging task to overcome for our future work.

5. CONCLUSION

In this paper, we have proposed a novel idea, an multiple path object detection network, to address the problem of object detection in resource constraint settings. The proposed deep detector is extremely light-weight and has a very competitive performance, which has been validated on the PASCAL VOC and COCO experiments.

Acknowledgement: The work was supported by NSFC 61876212, CCF-Tencent Open Research Fund and Hubei Scientific and Technical Innovation Key Project.

6. REFERENCES

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

[2] 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.

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

[4] Lichao Huang, Yi Yang, Yafeng Deng, and Yinan Yu, "Densebox: Unifying landmark localization with end to end object detection," *arXiv preprint arXiv:1509.04874*, 2015.

[5] Xinggang Wang, Kaibing Chen, Zilong Huang, Cong Yao, and Wenyu Liu, "Point linking network for object detection," *arXiv preprint arXiv:1706.03646*, 2017.

[6] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[7] Guotian Xie, Jingdong Wang, Ting Zhang, Jianhuang Lai, Richang Hong, and Guo-Jun Qi, "IGCV2: interleaved structured sparse convolutional neural networks," *CVPR*, vol. abs/1804.06202, 2018.

[8] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," *arXiv preprint arXiv:1807.11164*, 2018.

[9] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q Weinberger, "Multi-scale dense convolutional networks for efficient prediction," *CoRR, abs/1703.09844*, vol. 2, 2017.

[10] 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.

[11] Joseph Redmon and Ali Farhadi, "Yolo9000: better, faster, stronger," *arXiv preprint*, 2017.

[12] 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.

[13] Song Han, Huizi Mao, and William J Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[14] Jian-Hao Luo, Hao Zhang, Hong-Yu Zhou, Chen-Wei Xie, Jianxin Wu, and Weiyao Lin, "Thinet: Pruning cnn filters for a thinner net," *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[15] Gao Huang, Shichen Liu, Laurens van der Maaten, and Kilian Q Weinberger, "Condensenet: An efficient densenet using learned group convolutions," *group*, vol. 3, no. 12, pp. 11, 2017.

[16] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger, "Densely connected convolutional networks.," in *CVPR*, 2017, vol. 1, p. 3.

[17] Tsung-Yi Lin, Piotr Dollár, Ross B Girshick, Kaiming He, Bharath Hariharan, and Serge J Belongie, "Feature pyramid networks for object detection.," in *CVPR*, 2017, vol. 1, p. 4.

[18] Sergey Ioffe and Christian Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[19] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman, "The pascal visual object classes (voc) challenge," *International journal of computer vision*, vol. 88, no. 2, pp. 303–338, 2010.

[20] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[21] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. Ieee, 2009, pp. 248–255.

[22] 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.