The Summary of ”EfficientDet: Scalable and Efficient Object Detection” [1] Paper

Furkan Gül

I. Motivation

Recently, there has been a tremendous amount of research in making object detection tasks more accurate in the meantime keeping the computational cost as minimum as possible. As an example for one of the latest state-of-the-art (SOTA) object detector model, AmoebaNet-based NASFPN object detector [2] has about 167M parameters and 3045B FLOPs to manage SOTA accuracy. There are several real-world application areas for object detection which need light-weight models and low latency. Due to those two constraints, the research interest shifts to looking for models that have a smaller model size and cheaper computation costs for the inference. That’s why model efficiency becomes very crucial like a new metric for object detection tasks.

There have been many studies recently which made for developing higher efficient object detector model architectures such as one-stage type of models [3], [4] and compress previous models [5], [6]. Even though those models achieve higher efficiency, they generally lose some amount of model accuracy. So the critical question becomes is it possible to generate a scalable object detection model architecture with both better efficiency and higher accuracy across a high range of resource constraints like FLOPs constraint from 3B to 300? This paper tries to address and solve that problem by studying different design options for object detector architectures.

II. Approach

A. BiFPN

There is an crucial issue for object detection which is detecting features in scaled versions of an input image. To solve this problem, multi-scale feature fusion methodology is benefited to capture features at different resolutions. For this problem, the input is a list of multi-scale features $\overrightarrow{P}^{in} = (P_{l_1}^{in}, P_{l_2}^{in}, ...)$ where $P_{l_i}^{in}$ stands for the feature at level (layer) l_i. Here, the aims is to find a transformation f which effectively capture different features and give a list of new output features, $\overrightarrow{P}^{out} = f(\overrightarrow{P}^{in})$. As an example, Figure 1 shows a weighted bi-directional feature pyramid network (BiFPN) proposed in this paper. It gets level of 3-7 input features, $\overrightarrow{P}^{in} = (P_3^{in}, ..., P_7^{in})$. P_i^{in} denotes a feature level with $1/2^i$ resolution of the input images. As an example, if the resolution of the input image is 640x640, then P_3^{in} stands for the feature level 4 $(640/2^4 = 80)$ with resolution 80x80. The final feature level of 7, P_7^{in}, is represented with the resolution 5x5 $(640/2^7 = 5)$.

BiFPN has two-way information flows: a top-down and a bottom-up aggregation network as shown in Figure 1 as blue arrows and red arrows. These connections are called cross-scale connections. In this paper, they tried three main optimizations for cross-scale connections. The first optimization is removing nodes that only have just one input edge. Secondly, they add an extra edge from the input node to the output node if they both are at the same level. They also consider both the top-down and the bottom-up paths as one bidirectional path/one feature network layer. The third optimization for the cross-scale connections is that they repeat the same feature network layer multiple times to provide more high-level feature fusion. That’s why with all those three optimizations, they call their new feature network a bidirectional feature pyramid network (BiFPN) which can be seen in Figure 1 in detail. For combining/fusing features with different resolutions (from different feature levels), the most preferred method is to resize them to the same resolution and sum all these different features. All previous networks like pyramid attention network [7] and scaled version of it [8] process all input features equally without making any distinction among them. In this paper, since different input features are coming from different resolutions, they think that these input features most often contribute to the output feature unequally. To emphasize this problem, they recommend that the network itself must learn the importance of each input feature by adding additional weight for each input. For this additional weight, they propose a fast normalized fusion approach. In this approach, $O = \sum_i \frac{w_i}{\epsilon + \sum_j w_j} \cdot I_i$, where w_i is a learnable weight. It is a scalar for feature, a vector for channel, and a multi-dimensional tensor for pixel. By adding a Relu after each w_i, $w_i \geq 0$ is guaranteed. So the all weights are normalized probability from 0 to 1 which keeps the importance of each input. Since it is not a softmax, it is much more efficient in terms of computation cost. $\epsilon = 0.0001$ is standing for ensuring numerical stability.

In the final BiFPN architecture, the bidirectional cross-scale connections defined earlier and the fast normalized fusion combined together. Equations for the two fused features at the level of 6 in BiFPN (shown in Figure 1) are described below:

$$P_6^{td} = Conv(\frac{w_i \cdot P_6^{in} + w_2 \cdot Resize(P_7^{in})}{w_1 + w_2 + \epsilon}) \tag{1}$$

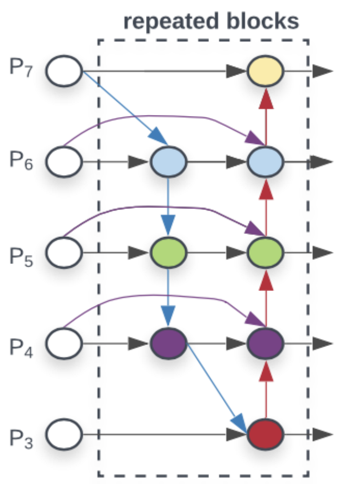

Fig. 1. BiFPN

$$P_6^{out} = Conv(\frac{w_i^{'} \cdot P_6^{in} + w_2^{'} \cdot P_6^{td} + w_3^{'} \cdot Resize(P_5^{out})}{w_1^{'} + w_2^{'} + w_3^{'} + \epsilon}) \tag{2}$$

Resize is generally a downsampling or upsampling process for resolution matching. *Conv* represents a convolutional operation for feature extraction. P_6^{td} represents the intermediate feature at the level of 6 on the top-down path. P_6^{out} represents the output feature at the level of 6 on the bottom-up path. All remaining features are built in the same methodology. To even get better efficiency, they use batch normalization and activation after each convolution operation and also add depthwise separable convolution for feature fusion [9].

B. *EfficientDet*

They proposed a new type of detection model named EfficientDet. In this network, EfficientNets [10] pre-trained on ImageNet [11] dataset are used as the backbone. Their proposed BiFPN structure is used as the feature network. It takes feature at the level 3-7, $(P_3, P_4, P_5, P_6, P_7)$, from the backbone network of EfficientNet. Then it applies the bidirectional feature fusion (consists of top-down and bottom-up paths) three times. These extracted fused features are given to a class network to predict object class and also are fed to a bounding box network to create a bounding box for predicted objects. All those processes can be seen in Figure 2.

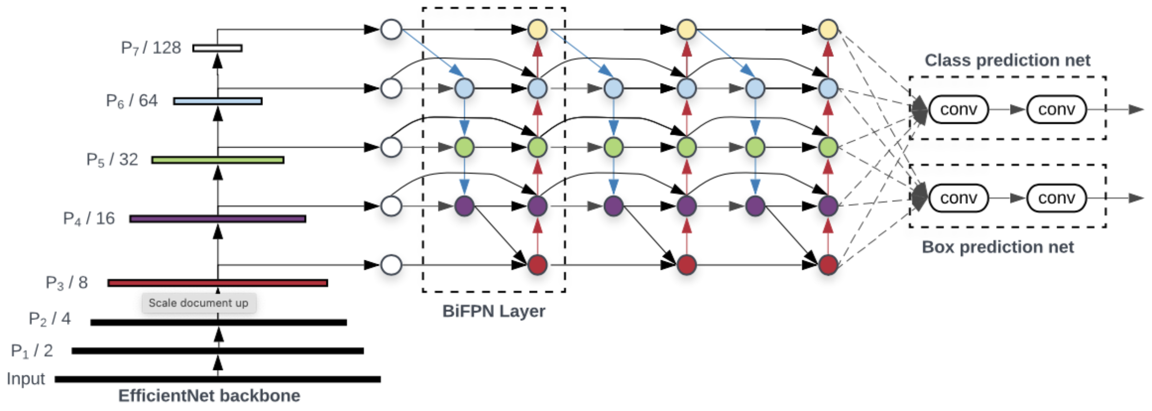

Fig. 2. EfficientDet Model Architecture

In order to scale up the baseline EfficientDet model, they use a basic compound coefficient, ϕ, to jointly scale-up all dimensions of backbone network, BiFPN, class prediction network, bounding box network, and resolution. For the backbone network, they use EfficientNet-B0 to B6 [10] with the same scaling coefficients.

For the BiFPN network, they linearly increase the depth of BiFPN network, D_{bifpn} (number of layers) and exponentially increase BiFPN width, W_{bifpn} (number of channels) by using Equation 3 below.

$$W_{bifpn} = 64 \cdot (1.35^{\phi}), D_{bifpn} = 3 + \phi \tag{3}$$

For the box and class prediction network, they fix the width as the same as the BiFPN and linearly increase the depth (number of layers) by using Equation 4 below.

$$W_{pred} = W_{bifpn}, D_{box} = D_{class} = \lfloor \phi/3 \rfloor \tag{4}$$

For the input image resolution, the input image resolution must be dividable by $2^7 = 128$. That's why they linearly increase resolutions using Equation 5 below.

$$R_{input} = 512 + \phi \cdot 128 \tag{5}$$

	Input size R_{input}	Backbone Network	BiFPN #channels W_{bifpn}	BiFPN #layers D_{bifpn}	Box/class #layers D_{class}
D0 ($\phi = 0$)	512	B0	64	3	3
D1 ($\phi = 1$)	640	B1	88	4	3
D2 ($\phi = 2$)	768	B2	112	5	3
D3 ($\phi = 3$)	896	B3	160	6	4
D4 ($\phi = 4$)	1024	B4	224	7	4
D5 ($\phi = 5$)	1280	B5	288	7	4
D6 ($\phi = 6$)	1280	B6	384	8	5
D7 ($\phi = 7$)	1536	B6	384	8	5
D7x	1536	B7	384	8	5

Fig. 3. Scaling configurations for EfficientDet D0-D6. ϕ is the compound coefficient that controls all other scaling dimensions; BiFPN, box/class networks, and input image resolution size are scaled up using equation 2, 3, 4 respectively.

	test-dev			val					Latency (ms)	
Model	AP	AP_{50}	AP_{75}	AP	Params	Ratio	FLOPs	Ratio	TitianV	V100
EfficientDet-D0 (512)	**34.6**	**53.0**	**37.1**	**34.3**	**3.9M**	**1x**	**2.5B**	**1x**	**12**	**10.2**
YOLOv3 [34]	33.0	57.9	34.4	-	-	-	71B	28x	-	-
EfficientDet-D1 (640)	**40.5**	**59.1**	**43.7**	**40.2**	**6.6M**	**1x**	**6.1B**	**1x**	**16**	**13.5**
RetinaNet-R50 (640) [24]	39.2	58.0	42.3	39.2	34M	6.7x	97B	16x	25	-
RetinaNet-R101 (640)[24]	39.9	58.5	43.0	39.8	53M	8.0x	127B	21x	32	-
EfficientDet-D2 (768)	**43.9**	**62.7**	**47.6**	**43.5**	**8.1M**	**1x**	**11B**	**1x**	**23**	**17.7**
Detectron2 Mask R-CNN R101-FPN [1]	-	-	-	42.9	63M	7.7x	164B	15x	-	56‡
Detectron2 Mask R-CNN X101-FPN [1]	-	-	-	44.3	107M	13x	277B	25x	-	103‡
EfficientDet-D3 (896)	**47.2**	**65.9**	**51.2**	**46.8**	**12M**	**1x**	**25B**	**1x**	**37**	**29.0**
ResNet-50 + NAS-FPN (1024) [10]	44.2	-	-	-	60M	5.1x	360B	15x	64	-
ResNet-50 + NAS-FPN (1280) [10]	44.8	-	-	-	60M	5.1x	563B	23x	99	-
ResNet-50 + NAS-FPN (1280@384)[10]	45.4	-	-	-	104M	8.7x	1043B	42x	150	-
EfficientDet-D4 (1024)	**49.7**	**68.4**	**53.9**	**49.3**	**21M**	**1x**	**55B**	**1x**	**65**	**42.8**
AmoebaNet+ NAS-FPN +AA(1280)[45]	-	-	-	48.6	185M	8.8x	1317B	24x	246	-
EfficientDet-D5 (1280)	**51.5**	**70.5**	**56.1**	**51.3**	**34M**	**1x**	**135B**	**1x**	**128**	**72.5**
Detectron2 Mask R-CNN X152 [1]	-	-	-	50.2	-	-	-	-	-	234‡
EfficientDet-D6 (1280)	**52.6**	**71.5**	**57.2**	**52.2**	**52M**	**1x**	**226B**	**1x**	**169**	**92.8**
AmoebaNet+ NAS-FPN +AA(1536)[45]	-	-	-	50.7	209M	4.0x	3045B	13x	489	-
EfficientDet-D7 (1536)	**53.7**	**72.4**	**58.4**	**53.4**	**52M**		**325B**		**232**	**122**
EfficientDet-D7x (1536)	**55.1**	**74.3**	**59.9**	**54.4**	**77M**		**410B**		**285**	**153**

We omit ensemble and test-time multi-scale results [30, 12]. RetinaNet APs are reproduced with our trainer and others are from papers.
‡Latency numbers with ‡ are from detectron2, and others are measured on the same machine (TensorFlow2.1 + CUDA10.1, no TensorRT).

Fig. 4. EfficientDet performance on MS COCO dataset. Results are for single-model single-scale. Latency is for inference with batch size 1. AA denotes auto-augmentation [2]. They group models together if they have similar accuracy, and compare their model size, FLOPs, and latency in each group [1].

They developed different architectures of EfficientNet-D0 ($\phi = 0$) to D7 ($\phi = 7$) by using Equation 3, 4, and 5 with different ϕ values. All detailts of EfficientNet architectures with seven different compound scaling coefficients (ϕ) proposed in this paper can be seen in Figure 3.

III. Dataset & Results

MSCOCO 2017 detection [12] dataset with 118K training images is used in their experiments to evaluate the suggested EfficientNet models. They use SGD optimizer with the momentum of 0.9 and weight decay of 4e-5 for all models. In the first epoch of training, the learning rate is linearly increased from 0 to 0.16. Then, it is reduced down using the cosine decay rule. They use SiLU activation [10] and exponential moving average with the decay of 0.9998.

In Figure 4, comparison between EfficientDet and other object detector can be seen. They report accuracy for both *test-dev* set which consists of 20k test images and *val* set which includes 5K validation images. By looking Figure 4, it can be concluded that their proposed EfficientDet models, are 4-9 times smaller and uses 13-42 times less FLOPs, have better efficiency than previous object detectors like YOLOv3 [3], RetinaNet-R40R101 [4], Detectron2 Mask R-CNN R101-FPN X101-FPN [13], ResNet-50 + NAS-FPN [8], AmoebaNet+ NAS-FPN +AA(1280) [2], Detectron2 Mask R-CNN X152 [13], and AmoebaNet+ NAS-FPN +AA(1536) [2].

IV. A Couple of Words

The next paper to be examined and summarized will be ”Auto-Encoding Scene Graphs for Image Captioning” [14] from 2018.

References

[1] Mingxing Tan, R. Pang, and Quoc V. Le. Efficientdet: Scalable and efficient object detection. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10778–10787, 2020.

[2] Barret Zoph, E. Cubuk, G. Ghiasi, Tsung-Yi Lin, Jonathon Shlens, and Quoc V. Le. Learning data augmentation strategies for object detection. In *ECCV*, 2020.

[3] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *ArXiv*, abs/1804.02767, 2018.

[4] Tsung-Yi Lin, Priyal Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42:318–327, 2020.

[5] Zhuang Liu, M. Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *ArXiv*, abs/1810.05270, 2019.

[6] Jonathan Pedoeem and R. Huang. Yolo-lite: A real-time object detection algorithm optimized for non-gpu computers. *2018 IEEE International Conference on Big Data (Big Data)*, pages 2503–2510, 2018.

[7] H. Li, Pengfei Xiong, Jie An, and Lingxue Wang. Pyramid attention network for semantic segmentation. In *BMVC*, 2018.

[8] G. Ghiasi, Tsung-Yi Lin, R. Pang, and Quoc V. Le. Nas-fpn: Learning scalable feature pyramid architecture for object detection. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7029–7038, 2019.

[9] F. Chollet. Xception: Deep learning with depthwise separable convolutions. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1800–1807, 2017.

[10] M. Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019.

[11] Jia Deng, W. Dong, R. Socher, L. Li, K. Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR 2009*, 2009.

[12] Tsung-Yi Lin, M. Maire, Serge J. Belongie, J. Hays, P. Perona, D. Ramanan, Piotr Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. *ArXiv*, abs/1405.0312, 2014.

[13] https://github.com/facebookresearch/ detectron2. 2020.

[14] X. Yang, Kaihua Tang, Hanwang Zhang, and J. Cai. Auto-encoding scene graphs for image captioning. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10677–10686, 2019.