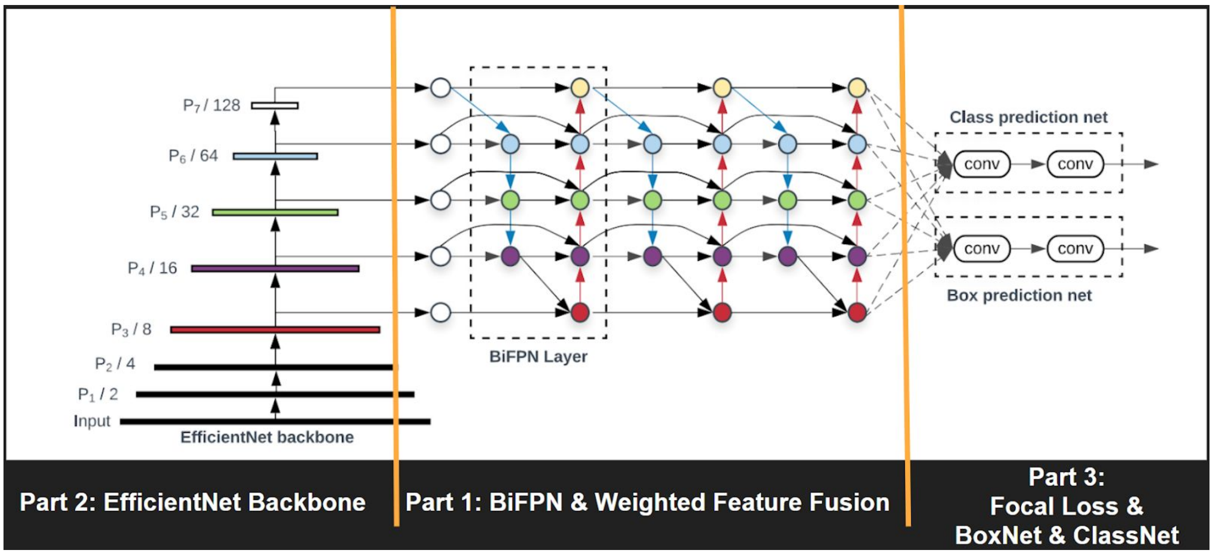
Competition 2 :Object Detection & Localization Report

Team members: 106033233 周聖諺
106033211 余孟旂
108062637 馮翔荏

1. Architecture

We use EfficientDet and here is the architecture overview.



2. BiFPN & Weighted Feature Fusion

Please refer to the paper: https://arxiv.org/abs/1911.09070

i. BiFPN(Bidirectional Feature Pyramid Network)

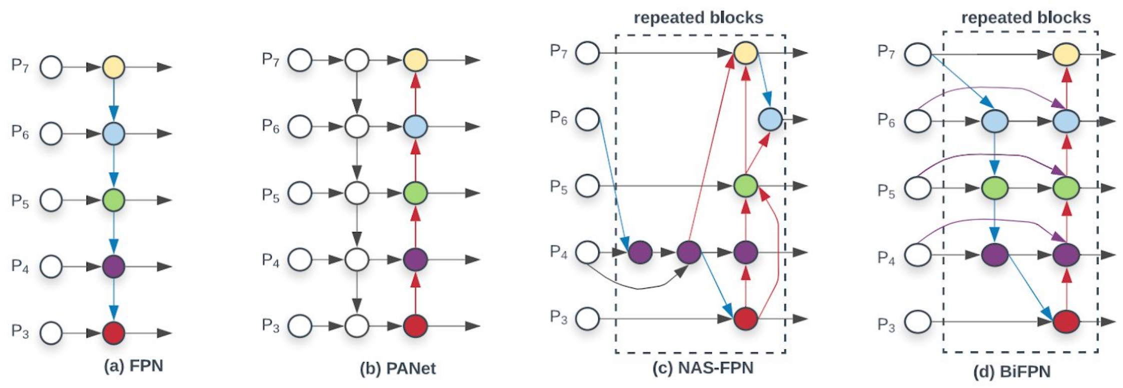
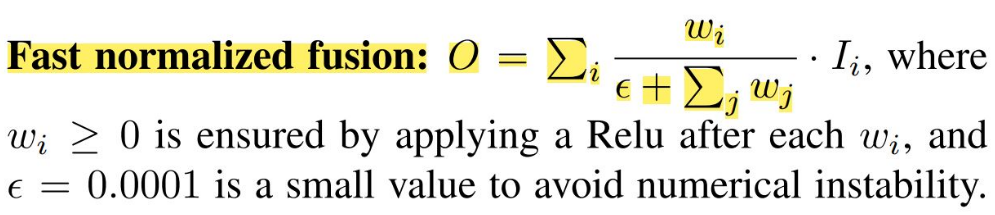


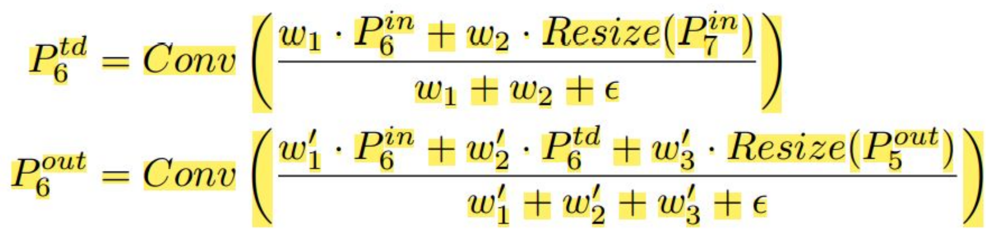
Figure 2: **Feature network design –** (a) FPN [23] introduces a top-down pathway to fuse multi-scale features from level 3 to 7 (P_3 - P_7); (b) PANet [26] adds an additional bottom-up pathway on top of FPN; (c) NAS-FPN [10] use neural architecture search to find an irregular feature network topology and then repeatedly apply the same block; (d) is our BiFPN with better accuracy and efficiency trade-offs.

It is inspired by the result of NAS-FPN. You can observe that NAS-FPN has a highway to connect the deeper layer of FPN and bidirectional propagation between different layers. As a result, the authors simplify the model of NAS-FPN and PANet and propose BiFPN. Unlike NAS-FPN, it doesn’t need to operate expensive architecture search but it can use the feature maps from different resolutions. Thus, it is good at detecting objects in different sizes.
For more detail, please refer to the function code model.py ‘build_wBiFPN_o(), SeparableConvBlock()’

ii. Weighted Feature Fusion

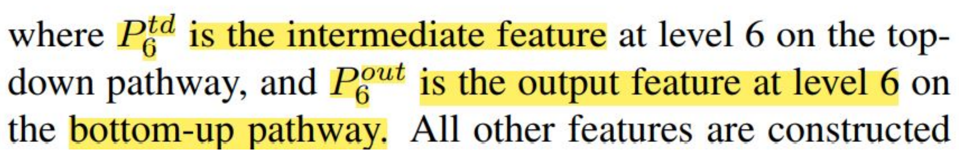


Fast normalized fusion: $O = \sum_i \frac{w_i}{\epsilon + \sum_j w_j} \cdot I_i$, where $w_i \geq 0$ is ensured by applying a Relu after each w_i, and $\epsilon = 0.0001$ is a small value to avoid numerical instability.



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

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



where P_6^{td} is the intermediate feature at level 6 on the top-down pathway, and P_6^{out} is the output feature at level 6 on the bottom-up pathway. All other features are constructed

Here we use fast normalized fusion which is the best result of the paper. Since the feature maps from different layers have different sizes, the EfficientDet first resizes the feature map to the target size with downsampling / upsampling and multiply a trainable weight to them. Then, apply the output to convolution, batchmormalize(with momentum 0.99 and epsilon 1e-3, refer to other implementation) and swish relu(the order is a little different from the original paper, since the paper doesn’t tell too much). Finally, propagate the result to the next layer.
For more detail, please refer to the function code model.py ‘wBiFPNAdd()’

iii. Training Epoch

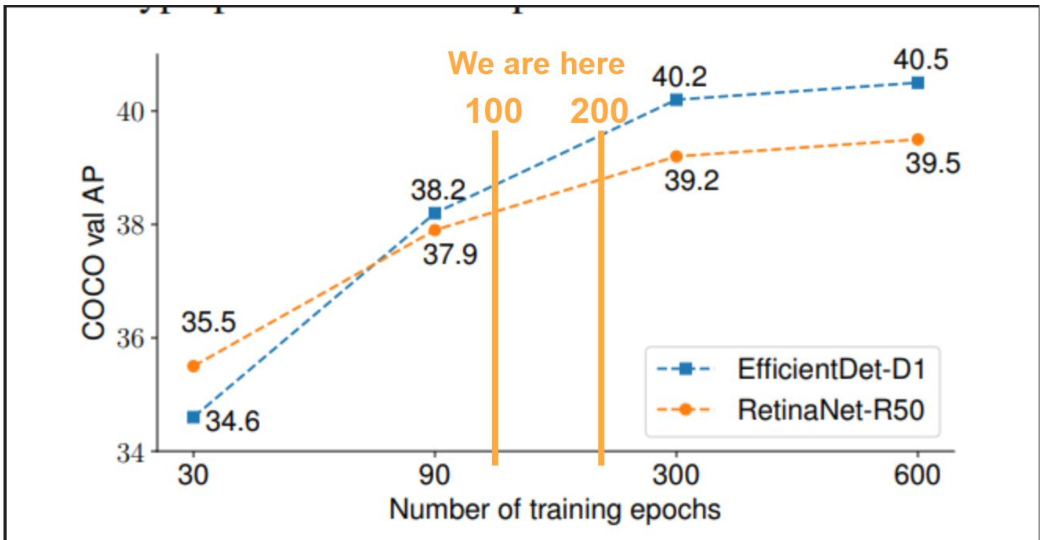
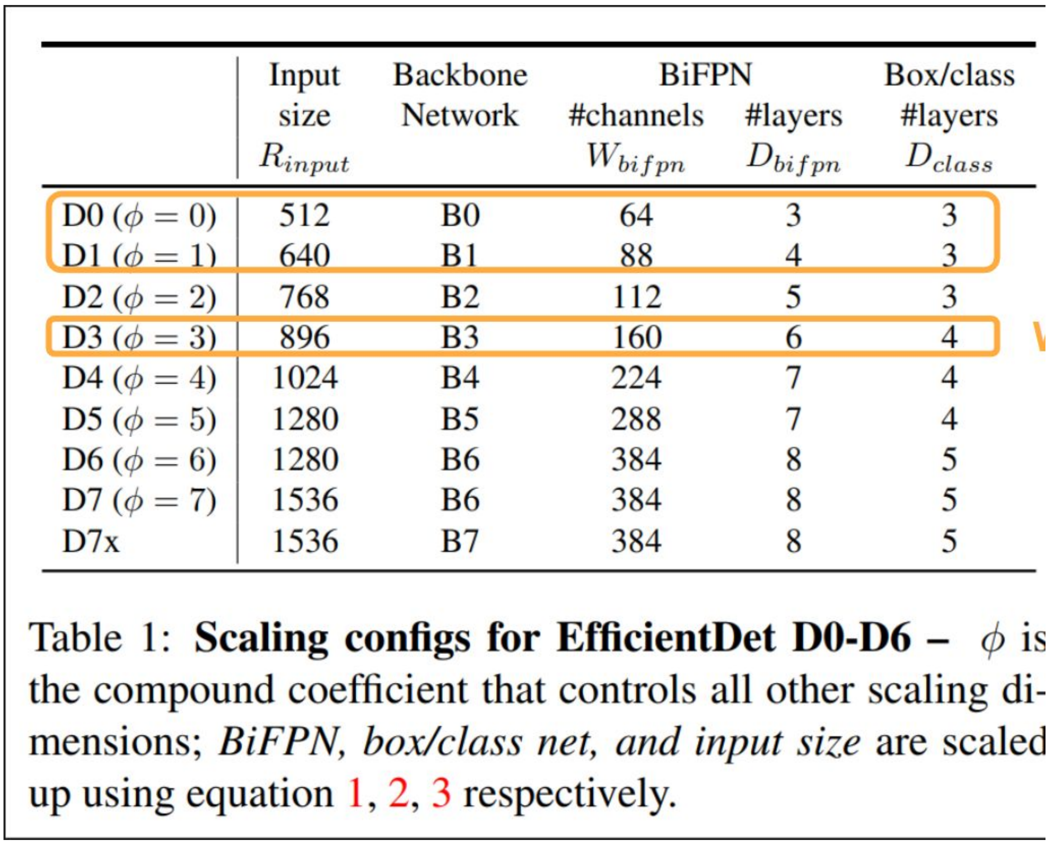


Figure 7: **Accuracy vs. Training Epochs.**

We train EfficientDet D3 for 100 ~ 200 epochs with learning rate 1e-3 and 1e-5(1e-3 is better). We freeze the backbone in the first 50 ~ 100 epochs and fine-tune in the remaining epochs. However, we observe that if we freeze the backbone, it is hard to reduce the loss after 50 epochs. In addition, there is a huge plateau in 0.6 loss and 0.47 loss.

iv. Model Parameters



	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

Table 1: **Scaling configs for EfficientDet D0-D6 –** ϕ is the compound coefficient that controls all other scaling dimensions; *BiFPN, box/class net, and input size* are scaled up using equation 1, 2, 3 respectively.

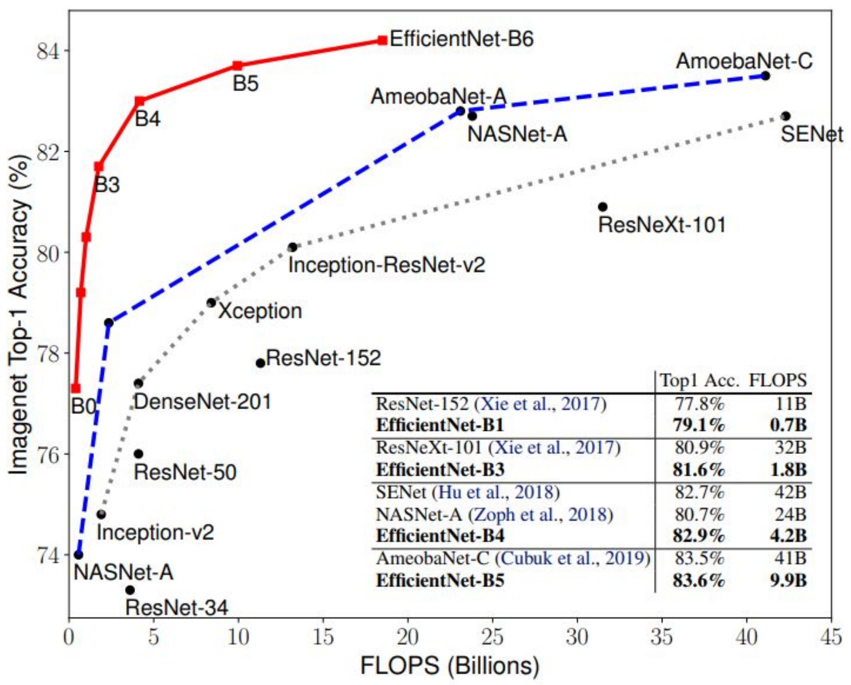
We use D0, D1, and D3. D3 has the best performance. D3 uses EfficientNet B3 as backbone, has BiFPN with 6 layers depth and BoxNet / ClassNet with 4 layers depth.

3. EfficientNet as Backbone

$$\begin{aligned}
\text{depth: } & d = \alpha^{\phi} \\
\text{width: } & w = \beta^{\phi} \\
\text{resolution: } & r = \gamma^{\phi} \\
\text{s.t. } & \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \\
& \alpha \geq 1, \beta \geq 1, \gamma \geq 1
\end{aligned}$$

EfficientDet model applies EfficientNet as its backbone, which efficiently scale the base model in different dimensions by factors of $\alpha^{\Phi}, \beta^{\Phi}\ \gamma^{\Phi}$. Φ is a parameter which controls how big you want to scale compared to the base model. In short, a bigger Φ means a bigger backbone. EfficientNet models can reach the same accuracy with lesser computation in comparison to other common-seen models like ResNet or Xception. As mentioned above, we trained models with Φ = 0,1 and 3. It turns out that a bigger model indeed performs better in our application.

One thing worth mention is that in the original paper, α, β, γ are found under the constraint $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$ using gridsearch.



	Top1 Acc.	FLOPS
ResNet-152 (Xie et al., 2017)	77.8%	11B
EfficientNet-B1	**79.1%**	**0.7B**
ResNeXt-101 (Xie et al., 2017)	80.9%	32B
EfficientNet-B3	**81.6%**	**1.8B**
SENet (Hu et al., 2018)	82.7%	42B
NASNet-A (Zoph et al., 2018)	80.7%	24B
EfficientNet-B4	**82.9%**	**4.2B**
AmeobaNet-C (Cubuk et al., 2019)	83.5%	41B
EfficientNet-B5	**83.6%**	**9.9B**

Figure 5. **FLOPS vs. ImageNet Accuracy –** Similar to Figure 1 except it compares FLOPS rather than model size.

For more detail, please refer to original paper: https://arxiv.org/abs/1905.11946

4. Focal Loss & L1-Smooth Loss & BoxNet & ClassNet

i. Focal loss

$$\mathrm{CE}(p, y) = \begin{cases} -\log(p) & \text{if } y = 1 \\ -\log(1-p) & \text{otherwise.} \end{cases} \quad (1)$$

$$p_t = \begin{cases} p & \text{if } y = 1 \\ 1-p & \text{otherwise,} \end{cases} \quad (2)$$

and rewrite $\mathrm{CE}(p, y) = \mathrm{CE}(p_t) = -\log(p_t)$.

$$\mathrm{CE}(p_t) = -\alpha_t \log(p_t). \quad (3)$$

$$\mathrm{FL}(p_t) = -(1-p_t)^\gamma \log(p_t). \quad (4)$$

$$\mathrm{FL}(p_t) = -\alpha_t(1-p_t)^\gamma \log(p_t). \quad (5)$$

We used focal loss for training ClassNet, as the paper wrote, one stage object detection model does not have region proposal network, so it cannot deal with the imbalance between foreground and background classes during training, and focal loss is a good idea to handle this problem. (1) is cross entropy for binary classification, (2) is a simple form for (1). An easy method for addressing class imbalance is to introduce a weighting factor, so it adds one factor in (3), but it just solves the imbalance of positive/negative examples and we also want to solve the imbalance of hard/easy examples. In (4), it adds a modulating factor $(1-p_t)^{\gamma}$ and (5) is the loss we used.

ii. L1-smooth loss

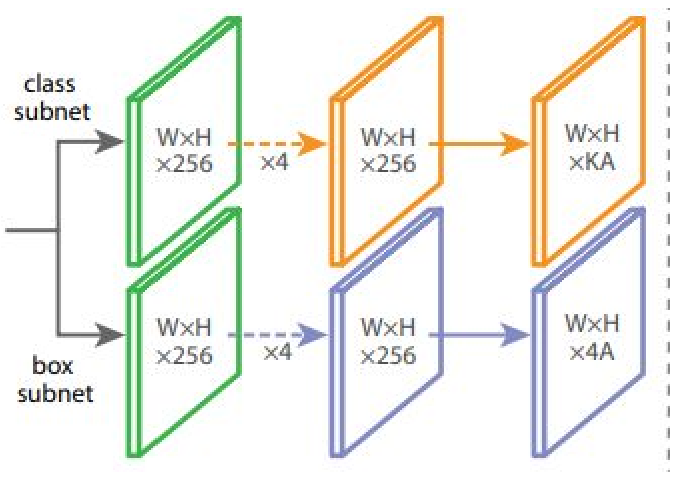
$$L_{\text{loc}}(t^u, v) = \sum_{i \in \{x,y,w,h\}} \text{smooth}_{L_1}(t_i^u - v_i),$$

in which

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise,} \end{cases}$$

L1-smooth loss is used for training BoxNet.It combines the advantages of L1-loss and L2-loss.When the absolute value of the argument is high It behaves as L1-loss,so the loss is not out of range.When the absolute value of the argument is closed to zero it behaves like L2-loss ,so it have less oscillations.

iii BoxNet & ClassNet



ClassNet is used to predict the probability of object presence at each spatial position for each of the A anchors and K object classes and output the KA binary predictions per spatial location .
BoxNet is used to predict the relative offset between the anchor and the groundtruth box.These two networks do not share parameters.