Appendix

1 Architecture of Label Encoding Function

Table 1. Architecture of our label encoding function. It has 19 layers. Most stages have the same output channels and stride as in *ResNet*-50 and *ResNet*-101, which is convenient for later supervision. Except that the first convolution has 80 and 128 for input and output channels respectively, instead of 3 and 64, in order to satisfy *COCO* dataset. We also remove the *max pooling* and we do not use *batch normalization*

Stage	**Block**	**Kernel Size**	**Stride**	**Output Channels**
Stage1	Conv	7×7	2	128
Stage2	ResBlock	1×1	1	64
		3×3	2	64
		1×1	1	256
Stage3	ResBlock	1×1	1	128
		3×3	2	128
		1×1	1	512
	ResBlock	1×1	1	128
		3×3	1	128
		1×1	1	512
Stage4	ResBlock	1×1	1	256
		3×3	2	256
		1×1	1	1024
	ResBlock	1×1	1	256
		3×3	1	256
		1×1	1	1024
Stage5	ResBlock	1×1	1	512
		3×3	2	512
		1×1	1	2048

2 Derivation of Eq. (7, 8)

In Sec. 1 and Sec. 3 we introduce our model as follows:

$$\theta_f^*, \theta_d^* = \arg\min_{\theta_f, \theta_d} \mathbb{E}_{(I,y)\sim\mathcal{D}} \mathcal{L}_{det}(d(f(I;\theta_f);\theta_d), y) + \lambda \mathcal{L}_{dis}(f(I;\theta_f), h(y;\psi^*)), \quad (1)$$

where the optimal weights ψ^* of the *label encoding function* ($h(\cdot)$) is derived from:

$$\psi^* = \arg\min_{\psi} \mathbb{E}_{(I,y)\sim\mathcal{D}} \mathcal{L}_{det}(d(h(y;\psi);\theta_d^*), y). \quad (2)$$

Clearly, there exist nested dependencies on the two variables θ_d^* and ψ^*. Thus the above equations are infeasible to compute directly.

Notice that in Eq. 1, ψ^* actually acts as a *constant* in the optimization. We define a function $\hat{\theta}_d(\cdot)$ as follows:

$$\hat{\theta}_d(\psi) \triangleq \arg\min_{\theta'_d} \left[\min_{\theta'_f} \mathbb{E}_{(I,y)\sim\mathcal{D}} \mathcal{L}_{det}(d(f(I;\theta'_f));\theta'_d), y) + \lambda \mathcal{L}_{dis}(f(I;\theta'_f), h(y;\psi)) \right]. \tag{3}$$

Compared with Eq. 1, we use θ'_d, θ'_f instead of θ_d and θ_f respectively for distinguishing. Easy to find that $\hat{\theta}_d(\psi^*) = \theta_d^*$. Then, we can rewrite Eq. 2 as follows:

$$\psi^* = \arg\min_{\psi} \mathcal{F}(\psi, \psi^*), \tag{4}$$

where

$$\mathcal{F}(\psi, \psi^*) = \mathbb{E}_{(I,y)\sim\mathcal{D}} \mathcal{L}_{det}(d(h(y;\psi); \hat{\theta}_d(\psi^*)), y). \tag{5}$$

Eq. 4 suggests that we need to find a certain ψ^* satisfying that the optimal point of the partial function $\mathcal{F}(\cdot, \psi^*)$ is also ψ^*, i.e. $\min_\psi \mathcal{F}(\psi, \psi^*) = \mathcal{F}(\psi^*, \psi^*)$. It motivates us to approximate ψ^* with the following optimization, since Eq. 4 is nontrivial to compute directly:

$$\begin{aligned} \psi^* &\simeq \arg\min_{\psi} \mathcal{F}(\psi, \psi) \\ &= \arg\min_{\psi} \mathbb{E}_{(I,y)\sim\mathcal{D}} \mathcal{L}_{det}(d(h(y;\psi); \hat{\theta}_d(\psi)), y), \end{aligned} \tag{6}$$

which derives our formulations in the text.

3 Feature Visualization

In this section we analyze our method with visualization on feature maps. We pick the second layer of the multi-scale feature maps from RetinaNet. We use images in validation set. We visualize each feature map with its intensity. Specifically, we use the $L2$ norm of each pixel. The larger the $L2$ norm is (i.e., the stronger the intensity), the brighter it is in the figure. Visualization results are shown in Fig. 1. Four columns are: (a) The original images. (b) Feature from baseline models. (c) Feature from our model. (d) Feature from our encoding function, which is the optimization target for (c). Compared with feature from baseline, which has clear boundary at the outline of each object, feature from ours is closer to boxes. Under the supervision from (d), the feature extends outside the object outline and "tries to reach the box edges". We believe this is beneficial for later instance extraction by detection head. Note that Fig. 1 is just a spatial projection of features. Information across channels is not visible here.

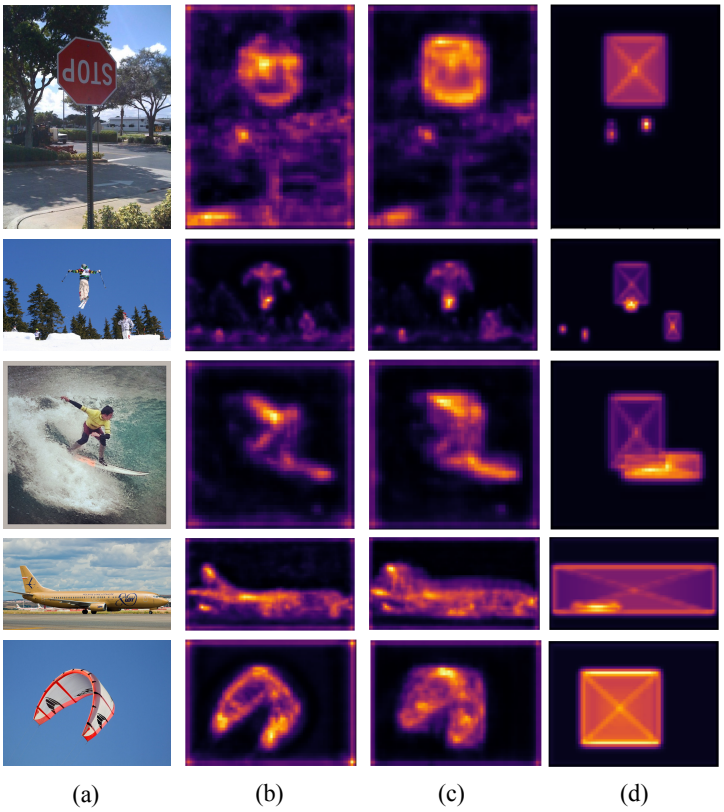

Fig. 1. Visualization of feature in baseline and ours. (a) The original images. (b) Feature from baseline. (c) Feature from ours. (d) Feature from our encoding function