A Appendix

A.1 Backbone Training and Inference

Following hyper-parameters and training settings provided by ResNet [26] and DenseNet [23], we pre-train ResNet-WS and DenseNet-WS networks on ImageNet ILSVRC without bells and whistles. This may place our models at a disadvantage. In details, we train ResNet-WS and DenseNet-WS for 120 epochs by 4 GPUs with 128 total batch size and 0.05 base learning rate. The architectures of pre-training on ImageNet ILSVRC is shown in Fig. A1. We use standard evaluation strategy for testing, which reports the error on the single 224×224 center crop from the images with a 256-pixel shorter side. ResNet18-WS and ResNet50-WS have 28.5% and 25.4% center-crop top-1 error, which is comparable to ResNet18 and ResNet50 with 30.24% and 24.7% top-1 error [26]. We find that the classification performance of ResNet50-WS is inferior to ResNet50. With more training tricks, *i.e.*, cosine learning rate decay, label smoothing, and mixup training, the performances of classification and final WSOD may be improved. DenseNet121-WS have 25.7% center-crop top-1 error, which is similar to DenseNet121 with 25.0% top-1 error [23].

A.2 Architectures of ResNet-WS for WSOD

Table A2 and A3 show detailed architectures of directly employing ResNet for WSOD task with C4 and C5 combinations, respectively. We also find that with only the top 1024 and 512 proposals, the performance increases from 26.5% to 27.8% and 28.8% mAP in C5, and from 30.3% to 31.1% and 32.6% mAP in C4 for WSDDN in ResNet50. As more object proposals may confuse the feature learning. In Table A4, we make more exposition of our each principle. Dilated convolutions are performed in conv4_x, and conv5_x with rate 2.

A.3 Space-Time Analysis

Directly using ResNet backbone with C4 combination for WSOD is highly complex. However, our ResNet-WS can largely help correct this problem. First, we use the C5 combination, which computes the proposed RAN block instead of the last stage of ResNet, to extract proposal features. Second, our experiments are mainly based on the F5 variant, which only needs to fine-tune the RAN block. With a small performance drop (about $1 \sim 2\%$ mAP on PASCAL VOC), F5 variant significantly reduces the computation complexity and training time. In our experiments, when training ResNet101-WS for WSDDN using GTX 1080Ti GPUs, each iteration takes 1.259 seconds for PASCAL VOC and 1.260 seconds for MS COCO. With 4 GPUs, it takes only 17.5 hours in PASCAL VOC for 40 epochs and 70 hours in MS COCO for 7 epochs. While vanilla ResNet101 with C4 combination takes about 2.03 seconds per iteration for F2 variant, and fails to converge for F5 variant without RAN.

Table A1: Architectures of ResNet-WS for image classification on ImageNet. Operators shown in braces are stacked with the numbers. Residual blocks are shown in brackets.

Layer	Output Size	ResNet18-WS	ResNet50-WS	ResNet101-WS
conv1_x	112×112	3×3, 64		
		2×2 MaxPool, stride 2		
		{ 3×3, 64 } × 2		
conv2_x	56×56	2×2 MaxPool, stride 2		
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	2×2 MaxPool, stride 2		
		$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$
conv4_x	14×14	2×2 MaxPool, stride 2		
		$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$
conv5_x	14×14	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
neck	7×7	2×2 MaxPool, stride 2		
	1×1	4096-d fc relu dropout 4096-d fc relu dropout	2048-d fc relu dropout 4096-d fc relu dropout	2048-d fc relu dropout 4096-d fc relu dropout
head	–	1000-d fc, softmax		

We further compare memory and time consumption with work in [43], which proposed sequential batch back-propagation to handle the large-memory usage of ResNet in WSOD. Follow [43], we used one input image per device, and change the number of proposals from $1k$ to $4k$ in $1k$ increments. Different to [43] that fixed the input image to be of size 600×600, we define the shortest side of input image to be of size 600. We use OICR+REG method [65] with ResNet101-WS, which has similar structures with [43]. The average training iteration time and memory consumption of our method are reported on 1080Ti GPUs. GPU memory usage is measured with Nvidia's system monitor interface (nvidia-smi), which outputs the overall GPU consumption of our detector system. As shown in Fig. A1, our ResNet101-WS has lower time and memory consumption.

Table A2: Architectures of directly adapting ResNet for WSOD with C4 combination.

Layer	Stride	ResNet18	ResNet50	ResNet101
conv1	2×2	7×7, 64, stride 2		
conv2_x	4×4	3×3 max pool, stride 2		
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	8×8	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$
conv4_x	16×16	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$
roi_extractor	–	7×7 RoIPool		
conv5_x	–	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
head	–	WSOD head		

Table A3: Architectures of directly adapting ResNet for WSOD with C5 combination.

Layer	Stride	ResNet18	ResNet50	ResNet101
conv1	2×2	7×7, 64, stride 2		
conv2_x	4×4	3×3 max pool, stride 2		
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	8×8	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$
conv4_x	16×16	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$
conv5_x	32×32	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
roi_extractor	–	7×7 RoIPool		
head	–	WSOD head		

Table A4: Architectures of ResNet-WS for WSOD.

Layer	Stride	ResNet18-WS	ResNet50-WS	ResNet101-WS
conv1_x	2×2	3×3, 64		
		2×2 MaxPool, stride 2		
		{ 3×3, 64 } × 2		
conv2_x	4×4	2×2 MaxPool, stride 2		
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	8×8	2×2 MaxPool, stride 2		
		$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$
conv4_x	8×8	2×2 MaxPool, stride 1		
		$\begin{bmatrix} 3\times3, 256, 2 \\ 3\times3, 256, 2 \end{bmatrix}\times2$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256, 2 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256, 2 \\ 1\times1, 1024 \end{bmatrix}\times23$
conv5_x	8×8	$\begin{bmatrix} 3\times3, 512, 2 \\ 3\times3, 512, 2 \end{bmatrix}\times2$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512, 2 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512, 2 \\ 1\times1, 2048 \end{bmatrix}\times3$
roi_extractor	–	7×7 RoIPool		
neck	–	4096-d fc relu dropout 4096-d fc relu dropout	2048-d fc relu dropout 4096-d fc relu dropout	2048-d fc relu dropout 4096-d fc relu dropout
head	–	WSOD head		

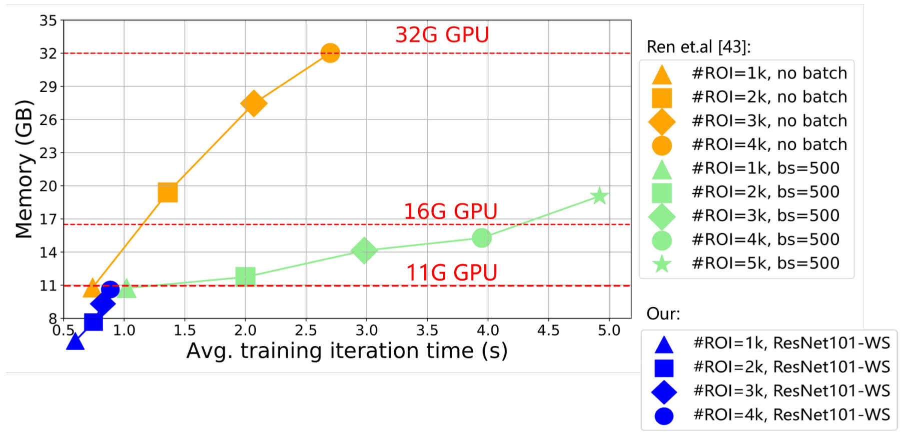

Fig. A1: ResNet-101 memory and time consumption using different methods and different number of proposals. We adapt this figure from the paper [43].